On the Power of the Randomized Iterate*

Iftach Haitner† Danny Harnik ‡ Omer Reingold§

May 5, 2011

Abstract

We consider two of the most fundamental theorems in Cryptography. The first, due to Håstad, Impagliazzo, Levin and Luby (STOC '89, STOC '90, SIAM J. on Computing '99), is that pseudorandom generators can be constructed from any one-way function. The second, due to Yao (FOCS '82), states that the existence of weak one-way functions implies the existence of full fledged one-way functions. These powerful plausibility results shape our understanding of hardness and randomness in Cryptography, but unfortunately their proofs are not as tight (i.e., security preserving) as one may desire.

This work revisits a technique that we call the *randomized iterate*, introduced by Goldreich, Drawback and Luby (SIAM J. on Computing '93). This technique was used by Goldreich et al. to give a construction of pseudorandom generators from regular one-way functions. We simplify and strengthen this technique in order to obtain a similar construction where the seed length of the resulting generators is as short as $\Theta(n \log n)$ rather than $\Theta(n^3)$ achieved by Goldreich et al. Our technique has the potential of implying seed-length $\Theta(n)$, and the only bottleneck for such a result are the parameters of current generators against space bounded computations. We give a construction with similar parameters for security amplification of regular one-way functions. This improves upon the construction of Goldreich, Impagliazzo, Levin, Venkatesan and Zuckerman (FOCS '90) in that the construction does not need to "know" the regularity parameter of the functions (in terms of security, the two reductions are incomparable). In addition, we use the randomized iterate to show a construction of a pseudorandom generator based on an exponentially-hard one-way function that has seed length of only $\Theta(n^2)$. This improves a recent result of Holenstein (TCC '06) that shows a construction with seed length $\Theta(n^5)$ based on such one-way functions. Finally, we show that the randomized iterate may even be useful in the general context of Håstad et al. In particular, we use the randomized iterate to replace the basic building block of the Håstad et al. construction. Interestingly, this modification improves efficiency by an $\Theta(n^3)$ factor and reduces the seed length to $\Theta(n^7)$ (which also implies improvement in the security of the construction).

Keywords: cryptography, pseudorandom generator, one-way functions, hardness amplification.

*This paper combines preliminary versions that appeared as [HHR06b] and [HHR06a].

†School of Computer Science, Tel Aviv University. E-mail: iftachh@cs.tau.ac.il. Research conducted while at Weizmann Institute of Science. Research supported by grant no. 1300/05 from the Israel Science Foundation.

‡IBM Haifa Research Labs. danny.harnik@gmail.com. Research conducted while at the Weizmann Institute and the Technion.

§Microsoft Research, Silicon Valley Campus, and Weizmann Institute of Science. E-mail:omreing@microsoft.com. Supported by grant no. 1300/05 from the Israel Science Foundation.

Contents

1 Introduction

In this work we address two fundamental problems in cryptography: (1) constructing pseudorandom generators from one-way functions and (2) transforming weak one-way functions into strong one-way functions. The common thread linking the two problems in our discussion is the technique we use. This technique that we call the *randomized iterate* was introduced by Goldreich, Krawczyk and Luby [GKL93] in the context of constructing pseudorandom generators from regular one-way functions. We revisit this method, simplify existing proofs and utilize our new perspective to achieve significantly better parameters for security and efficiency. We demonstrate that the randomized iterate is also applicable to the construction of pseudorandom generators *from any one-way function*. Specifically, we revisit the seminal paper of Håstad, Impagliazzo, Levin and Luby [HILL99] and show that the randomized iterate can help improve the parameters in this context. We also give significant improvements to the construction of pseudorandom generators from one-way functions that are exponentially hard to invert. Finally, we use the randomized iterate both to simplify and to strengthen previous results regarding efficient hardness amplification of regular one-way functions.

We start by introducing the randomized iterate in the context of pseudorandom generators, and postpone the discussion on amplifying weak to strong one-way function to Section 1.3.

1.1 Pseudorandom Generators and the Randomized Iterate

Pseudorandom generators, first introduced by Blum and Micali [BM82], and stated in its current, equivalent form, by Yao [Yao82], are one of the cornerstones of cryptography. Informally, a pseudorandom generator is a polynomial-time computable function G that stretches a short random string x into a long string $G(x)$ that "looks" random to any efficient (i.e., polynomial-time) algorithm. Hence, there is no efficient algorithm that can distinguish between $G(x)$ and a truly random string of length $|G(x)|$ with more than a negligible probability. Originally introduced in order to convert a small amount of randomness into a much larger number of effectively random bits, pseudorandom generators have since proved to be valuable components for various cryptographic applications such as bit commitments [Nao91], pseudorandom functions [GGM86], and pseudorandom permutations [LR88], to name a few.

1.1.1 Previous constructions

The first construction of a pseudorandom generator was given in [BM82] based on a particular one-way function and was later generalized in [Yao82] into a construction of a pseudorandom generator based on any one-way permutation (hereafter, the BMY construction). The BMY generator works by iteratively applying the one-way permutation on its own output. More precisely, for a given function f and input x define the k'th iterate recursively as $f^k(x) = f(f^{k-1}(x))$, where $f^1(x) = f(x)$. To complete the construction, one needs to take a hardcore-bit at each iteration. If we denote by $b(x)$ the hardcore-bit of x (take for instance the Goldreich-Levin [GL89] predicate), then the BMY generator on seed x outputs the hardcore-bits $b(f^1(x)), \ldots, b(f^m(x))$.[1]

The natural question arising from the BMY generator was whether one-way permutations are actually necessary for pseudorandom generators, or can one do with a more relaxed notion. Specifically, is any one-way function sufficient for pseudorandom generators? Levin [Lev87] observed that

[1] We mention that typically the BMY generator is presented as $b(x), b(f^1(x)), \ldots, b(f^{m-1}(x))$. For consistency with our results, however, we present it so that the first hardcore bit is taken *after* the first iteration.

the BMY construction works for any *one-way function on its iterates*, that is, a one-way function that remains one-way when applied sequentially on its own outputs. A general one-way function, however, does not have this property since the output of f may have very little randomness in it, and a second application of f may be easy to invert. A partial solution was suggested by Goldreich et al. [GKL93] that showed a construction of a pseudorandom generator based on any regular one-way function (hereafter, the GKL generator). A regular function is a function such that every element in its image has the same number of preimages. The GKL generator introduced the technique at the core of this work, that we call the *randomized iterate*. Rather than simple iterations, an extra randomization step is added between every two applications of f. More precisely,

Definition 1.1 (the randomized iterate (informal))**.** *For function f, input x and random vector of hash functions $\overline{h} = (h_1, \ldots, h_m)$, recursively define the k'th randomized iterate (for $2 \leq k \leq m+1$) by:*

$$f^k(x, \overline{h}) = f(h_{k-1}(f^{k-1}(x, \overline{h}))),$$

where $f^1(x, \overline{h}) = f(x)$.

The rational is that $h_k(f^k(x, \overline{h}))$ is now uniformly distributed, and the challenge is to show that f, when applied to $h_k(f^k(x, \overline{h}))$, is hard to invert even when the randomizing hash functions $\overline{h} = (h_1, \ldots, h_m)$ are made public. Once this is shown, the generator is similar in nature to the BMY generator (the generator outputs $b(f^1(x, \overline{h})), \ldots, b(f^m(x, \overline{h})), \overline{h}$).

Finally, Håstad et al. [HILL99] (combining [ILL89, Hås90]), culminated this line of research by showing a construction of a pseudorandom generator using any one-way function (hereafter called the HILL generator). This result is one of the most fundamental and influential theorems in cryptography. It introduced many new ideas that have since proved useful in other contexts, such as the notion of pseudoentropy, and the implicit use of family of pairwise-independent hash functions as randomness extractors. We mention that HILL departs from GKL in its techniques, taking a significantly different approach.

1.1.2 The complexity and security of the previous constructions

While the HILL generator fully answers the question of the plausibility of a generator based on any one-way function, the construction is highly involved and very inefficient. Other than the evident contrast between the simplicity and elegance of the BMY generator to the complex construction and proof of the HILL generator, the parameters achieved in the construction are far worse, rendering the construction impractical.

In practice, it is not necessarily sufficient that a reduction translates polynomial security into polynomial security. In order for reductions to be of any practical use, the concrete overhead introduced by the reduction comes into play. There are various factors involved in determining the security of a reduction, and in Section 2.10 we elaborate on the security of cryptographic reductions and the classification of reductions in terms of their security. Here, however, we focus only on one central parameter, which is the length d of the generator's seed compared to the length n of the input to the underlying one-way function. The BMY generator takes a seed of length $\Theta(n)$, the GKL generator takes a seed of length $\Theta(n^3)$ while the HILL construction produces a generator with seed length on the order of $\Theta(n^8)$.[2]

[2] The seed length actually proved in [HILL99] is $\Theta(n^{10})$, however it is mentioned that a more careful analysis can get to $\Theta(n^8)$. A formal proof for the $\Theta(n^8)$ seed length construction is given by Holenstein [Hol06a].

The length of the seed is of great importance to the security of the resulting generator. While it is not the only parameter, it serves as a lower bound to how good the security may be. For instance, the HILL generator on d bits has security that is at best comparable to the security of the underlying one-way function, but only on $\Theta(\sqrt[8]{d})$ bits. To illustrate the implications of this deterioration in security, consider the following example: suppose that we only trust a one-way function when applied to inputs of at least 100 bits, then the GKL generator can only be trusted when applied to a seed of length of at least one million bits, while the HILL generator can only be trusted on seed lengths of 10^{16} and up (both being highly impractical). Thus, trying to improve the seed length towards a linear one (as it is in the BMY generator) is of great importance in making these constructions practical.

1.1.3 Exponentially hard one-way functions and improving the seed length

The BMY and GKL generators demonstrate that assuming restrictions on the underlying one-way function allows for great improvement of the seed length (or input blowup). The common theme in these restrictions is that they deal with the *structure* of the one-way function. A different approach was recently taken by Holenstein [Hol06a], who builds a pseudorandom generator from any one-way function with *exponential hardness*, i.e., for some constant c, no algorithm of running time at most 2^{cn} inverts the function with probability better than 2^{-cn}. This approach is different as it discusses *raw hardness* as opposed to structure. The result in [Hol06a] is essentially a generalization of the HILL generator that also takes into account the parameter stating the hardness of the one-way function. In its extreme case where the hardness is exponential, the pseudorandom generator takes a seed length of $d = \Theta(n^5)$ and has security $2^{\Theta(d^{\frac{1}{5}})}$. The seed length can be reduced to as low as $\Theta(n^4 \log^2 n)$ when the resulting generator is only required to have super-polynomial security (i.e. security of $n^{\log n}$). In its other extreme based on a general one-way function (with super-polynomial hardness), [Hol06a] forms a formal proof of the best known seed length for the HILL construction (seed length $\Theta(n^8)$).

1.2 Our Results on Pseudorandom Generators

Our improvements to the seed length of pseudorandom generators under the various assumptions are summarized in Figure 1. In the upcoming section we elaborate on each of these constructions and highlight the source of the improvements.

Paper	Type of function	Seed length
[BM82, Yao82]	One-way permutation	$\Theta(n)$
[GKL93]	Regular one-way function	$\Theta(n^3)$
This work		$\Theta(n \log n)$
[Hol06a]	One-way function with exponential hardness	$\Theta(n^5)$
This work		$\Theta(n^2)$
This work	Regular one-way function with exponential hardness	$\Theta(n)$
[HILL99, Hol06a]	Any one-way function	$\Theta(n^8)$
This work		$\Theta(n^7)$

Figure 1: Summary of results.

1.2.1 Regular one-way functions

We give a construction of a pseudorandom generator from any regular one-way function with seed length $\Theta(n \log n)$. We mention that our approach has the potential of reaching a construction with a linear seed, the bottleneck being the efficiency of the currently known bounded-space generators. Our construction follows the randomized iterate method and is achieved in two steps:

- We give a significantly simpler proof that the GKL generator works, allowing the use of a family of hash functions that is pairwise-independent rather than n-wise independent (as used in [GKL93]). This gives a construction with seed length $\Theta(n^2)$ (see Theorem 3.6).
- The new proof allows for the derandomization of the choice of the randomizing hash functions via the *generator against bounded-space adversaries* (for short, bounded-space generator) of Nisan [Nis92], further reducing the seed length to $\Theta(n \log n)$ (see Theorem 3.10).

The proof method. Following is a high-level description of our proof method. For simplicity we focus on the second randomized iteration (i.e., on $f^2(x,h) = f(h(f(x)))$, but the same argument generalizes to the other iterations. The main task at hand is to show that it is hard to find $f^1(x,h) = f(x)$ when given $f^2(x,h)$ and h. This follows by showing that any procedure A for finding $f(x)$ given $(f^2(x,h),h)$ enables to invert the one-way function f (on a random image). Specifically, we show that for a random image $z \in f(U_n)$, if we choose a random and independent hash h' and feed the pair (z,h') to A, then A is likely to return a value $f(x')$ such that $h'(f(x')) \in f^{-1}(z)$ (and thus we obtain an inverse of z). This is ultimately shown by proving that if A succeeds on the distribution of $(f^2(x,h),h)$, then A is also successful on the distribution of $(f^2(x,h),h')$, where h' is chosen independently of (x,h).

Our proof is inspired by a technique used by Rackoff in his proof of the Leftover Hash Lemma (in [IZ89]). Rackoff proves that a distribution is close to uniform by showing that it has *collision probability* that is very close to that of the uniform distribution.[3] We would have liked to follow this scheme and consider the collision probability of the two aforementioned distributions. In our case, however, the two distributions could actually be very far from each other. Yet, based on our analysis of the collision-probabilities we manage to prove that the probability of any event under the first distribution is *polynomially related* to the probability of the same event under the second distribution. This proof generalizes nicely also to the case of many iterations.

The derandomization using a bounded-space generator follows directly from the new proof. The point is to introduce a derandomization of the hash functions such that the collision probability of the randomized iterate remains essentially the same. Since the proof centers around the collision probability of $(f^m(x,\overline{h}),\overline{h})$, the proof will hold also for the derandomized version. More precisely, consider the procedure that given inputs x_0, x_1 and $\overline{h} = (h_1,\ldots,h_{m-1})$, outputs '1' if $f^m(x_0,\overline{h})$ equals $f^m(x_1,\overline{h})$ and '0' otherwise. Note that the probability, over a uniform choice of inputs, that the above procedure outputs '1', is exactly the collision probability of $(f^m(x,\overline{h}),\overline{h})$. Also note that the above procedure can run in linear space, since it simply needs to store the two intermediate iterates at each step. Therefore, the probability that the above procedure outputs '1' while replacing $\overline{h}$ with the output of a generator against linear space adversaries, is very close to the collision probability of $(f^m(x,\overline{h}),\overline{h})$. It follows that the collision probability of $(f^m(x,\widetilde{h}),\widetilde{h})$,

[3] The collision probability of a distribution is the probability of getting the same element twice when taking two independent samples from the distribution.

where $\widetilde{h}$ is now the output of the bounded-space generator is very close to that of $(f^m(x,\overline{h}),\overline{h})$, and the security proof now follows as in the proof when using independent randomizing hash functions. We mention that derandomization of similar spirit was used by Phillips [Phi92], in his efficient amplification of weak one-way permutations (see Section 1.3).

1.2.2 Exponentially hard one-way functions

We give a construction of a pseudorandom generator from *any* exponentially-hard one-way function with seed length $d = \Theta(n^2)$ and security $2^{\Theta(\sqrt{d})}$. If we only require the security of the resulting generator to be super-polynomial, then the construction gives seed that is only $\Theta(n \log^2 n)$ long. We mention that Holenstein's result applies for *any* one-way function (but is most efficient when the one-way function is exponentially hard). Our construction on the other hand is specialized for one-way functions with exponential hardness, and does not generalize to use significantly weaker one-way functions. More concretely, our construction can use any one-way function with security $2^{\phi(n)}$, *as long as* $\phi(n) \in \Omega(\frac{n}{\log n})$.

The core technique of our construction is once again the randomized iterate. Trying to apply the randomized iterate to a general one-way function, we encounter the following difficulty: for $k \geq 2$, the k'th randomized iteration of a general one-way function may be easy on a large fraction of the inputs. Our key observation is that the randomized iterate cannot be easy everywhere. Our Lemma 4.1 indicates that for every one-way function f, there exists a set $\mathcal{S}$ of inputs to f^k such that the k'th randomized iteration is hard to invert over inputs taken from this set. Moreover, the density of $\mathcal{S}$ is at least $\frac{1}{k}$. This means that there is some pseudorandomness to be extracted from the k'th randomized iterate; taking a hard-core bit of the k'th randomized iteration gives a bit that with probability $\frac{1}{k}$ looks random (to a computationally bounded observer). Our idea is to collect these bits that contain some pseudoentropy and to then extract from them the pseudorandom output of the generator.

Consider taking t independent copies of the randomized iterate (on t independent inputs) and for each of the t copies taking a hardcore bit from the k'th iteration. This forms a string of t bits, of which $\frac{t}{k}$ are expected to be random looking. Our next step would be to run a *randomness extractor* on this string, to generate $\Theta(\frac{t}{k})$ pseudorandom bits. The problem, however, is that the total number of pseudorandom bits generated, i.e., $\Theta(\sum_{k=1}^{m} \frac{t}{k})$, is too low, and in particular insufficient to compensate for the tn bits invested in the random seed.

This problem can be remedied by taking more hardcore bits at each iteration. Specifically, if the one-way function has exponential hardness then a linear number of hardcore bits may be taken at each iteration (Goldreich and Levin [GL89]). Thus, taking $t = n$ independent copies, the total number of pseudorandom bits generated can be larger than the seed length. The construction gives a seed that is $\Theta(n^2)$-long, as each independent copy of the randomized iterate only runs a constant number of iterations.

Remark 1.2 (on randomness extractors and pseudorandomness)**.** *The use of randomness extractors in a computational setting, was initiated in [HILL99]. We give a general "uniform extraction lemma" (Lemma 2.14) for this purpose that is proved using a uniform hardcore Lemma of Holenstein from [Hol05]. We mention that a similar proof was given independently in [Hol06a].*

1.2.3 Any one-way function

The HILL generator takes a totally different path than the GKL generator. The initial step in the HILL construction takes a one-way function f and generates a bit that has significantly more *pseudoentropy* than actual entropy. This gap is then exploited in order to build a full-fledged pseudorandom generator. This initial construction does not use iterations of f at all. We ask whether the technique of randomized iterations can be helpful for the case of any one-way function, and give a positive answer to this question (actually, we are using only the first two iterations). Specifically, this method also improves the efficiency of the overall construction by an $\Theta(n^3)$ factor (ignoring polylog(n) terms) over the original HILL generator and reduces the seed length by a factor of n, which also implies improvement in the security of the construction. All in all, we present (Theorem 5.7) a pseudorandom generator from *any* one-way function with seed length $\Theta(n^7)$.

Our generator replaces the initial step of the HILL generator with a different construction based on the techniques we have developed. We briefly describe the new initial step. Denote the degeneracy of y by $\mathrm{D}_f(y) = \left\lceil \log \left| f^{-1}(y) \right| \right\rceil$ (this is a measure that divides the images of f to n categories according to their preimage size). Let b denote a hardcore-bit (again we take the Goldreich-Levin hardcore-bit [GL89]). Loosely speaking, we consider the bit $b(f(x))$ when given the value $(f^2(x,h),h)$ and make the following observation: when $\mathrm{D}_f(f(x)) \geq \mathrm{D}_f(f^2(x,h))$, the value $b(f(x))$ is (almost) fully determined by $(f^2(x,h),h)$; as opposed to when $\mathrm{D}_f(f(x)) < \mathrm{D}_f(f^2(x,h))$, where no information about $b(f(x))$ leaks. But in addition, if $\mathrm{D}_f(f(x)) = \mathrm{D}_f(f^2(x,h))$, then $b(f(x))$ is computationally-indistinguishable from uniform (that is, looks uniform to any efficient observer), even though it is actually fully determined. The latter stems from the fact that when $\mathrm{D}_f(f(x)) = \mathrm{D}_f(f^2(x,h))$ the behavior is close to that of a regular function.

As a corollary we get that the bit $b(f(x))$ has entropy of no more than $\frac{1}{2}$ (i.e., the probability of $\mathrm{D}_f(f(x)) < \mathrm{D}_f(f^2(x,h))$), but has "entropy of at least $\frac{1}{2} + \frac{1}{\Theta(n)}$ in the eyes of any computationally-bounded observer" (i.e., the probability of $\mathrm{D}_f(f(x)) \leq \mathrm{D}_f(f^2(x,h))$). In other words, $b(f(x))$ has entropy $\frac{1}{2}$ but pseudoentropy of $\frac{1}{2} + \frac{1}{\Theta(n)}$.[4] As in HILL, it is this gap of $\frac{1}{\Theta(n)}$ between the entropy and pseudoentropy that eventually allows the construction of a pseudorandom generator.

Comparing to HILL. The HILL construction builds a pair of function and predicate such that the predicate has entropy p, but pseudoentropy of at least $p + \frac{1}{\Theta(n)}$ (see Appendix A for the description of their pair). Unlike in our construction, however, the entropy threshold p in the HILL construction is unknown (i.e., not efficiently computable). This is a real disadvantage, since knowledge of this threshold is essential for the overall construction. To overcome this, the HILL generator enumerates all values for p (up to an accuracy of $\Theta(\frac{1}{n})$), runs the generator with each of these values and eventually combines all generators using an XOR of their outputs. This enumeration costs an additional factor n to the seed length as well an additional factor of n^3 to the number of calls to the underlying function f, and hence our efficiency and security improvements.

Remark 1.3. *[on pseudorandomness in* NC^1*] For the most part, the HILL construction is "depth" preserving. In particular, given two "non-uniform" hints of* $\log n$ *bits each (that specify two different*

[4] It natural to ask why should we consider $b(f(x))$ as the predicate and not simply $b(x)$. Clearly the pseudoentropy of $b(x)$, given $(f^2(x,h),h)$ is at least as large as that of $b(f(x))$ (since $f(x)$ is determined by x). The problem is that the real entropy of $b(x)$ in this case is unknown (and may, in fact, be as high as the pseudoentropy). In other worlds, when considering $b(f(x))$ rather than $b(x)$, we reduce the conditional entropy to a known bound, while keeping the pseudoentropy larger than this bound.

properties of the one-way function [5]*), the reduction gives generators in* NC^1 *from any one-way function in* NC^1*. Unfortunately, without these hints, the depth of the construction is polynomial (rather than logarithmic). Our construction eliminates the need for one of these hints (we still need to know the entropy of the function) and thus can be viewed as a step towards achieving generators in* NC^1 *from any one-way function in* NC^1*. Building pseudorandom generators in* NC^1 *(from one-way functions in* NC^1*) would be highly significant. In particular, since [AIK06] showed that such generators imply pseudorandom generators in* NC^0*.*

1.2.4 The recent generator of Haitner, Reingold and Vadhan

In a very recent result, Haitner, Reingold and Vadhan [HRV10] presented new family of pseudorandom generator for "any hardness". In particular, starting from standard (polynomially secure) one-way functions, their construction achieves seed length $\Theta(n^4)$ (to compare with the $\Theta(n^7)$ achieved here), where when starting from exponentially-hard one-way functions, their seed length matches the result presented here.[6] Additional advantage of the generators of [HRV10] over this work (and over [HILL99, Hol06a]), is that their generators use non-adaptive calls to the underlying one-way functions. In particular, their results yields that pseudorandom generators in NC^0 can be constructed from one-way functions in NC^1 (see Remark 1.3).

1.3 One-Way Functions – Amplification from Weak to Strong

The existence of one-way functions is essential to almost any task in cryptography (see for example [IL89]) and also sufficient for numerous cryptographic primitives such as the pseudorandom generators discussed above. In general, for constructions based on one-way functions we use what are called *strong* one-way functions. That is, functions that can only be inverted efficiently with negligible success probability. A more relaxed definition is that of a ε-weak one-way function, where $\varepsilon = \varepsilon(n)$ is a polynomial fraction. This is a function that every efficient algorithm fails to invert on at least a ε fraction of the inputs. This definition is significantly weaker, yet, Yao [Yao82] showed how to convert any weak one-way function into a strong one (see proof in [Gol01]). The new strong one-way function simply consists of many independent copies of the weak function concatenated to each other. The solution of Yao, however, incurs a blow-up factor of $\omega(\log(n)/\varepsilon)$ to the input length of the strong function, which translates to a significant loss in the security (as in the case of pseudorandom generators).

Goldreich et al. [GIL+90] pointed out this loss of security problem and gave a solution for one-way permutations that has just a linear blowup in the length of the input. Their solution was also generalized to *known-regular* one-way functions (regular functions whose image size is efficiently computable), where its input length varied according to the required security. The input length is linear when the required security is $2^{O(\sqrt{n})}$, but deteriorates up to $\Theta(n^2)$ when the required security is higher (e.g., security $2^{\Theta(n)}$).[7] Their construction uses a variant of randomized iterates, where the randomization is via one random step on an expander graph. Additional attempts to avoid this loss of security problem were given by [Phi92, DI99] (see below).

[5] Consider the random variable induced by applying the one-way function on a uniformly chosen input. One of these hints relates to the entropy of this random variable, where the other hint, p, relates to its variance.

[6] Their result also generalizes to functions with in between hardness, significantly improving over [Hol06a] for every choice of hardness.

[7] Loosely speaking, one can think of the security as the probability of finding an inverse to a random image $f(x)$ simply by choosing a random element in the domain.

1.3.1 Our Contribution to hardness amplification

We present an alternative efficient hardness amplification for regular one-way functions. Specifically, in Theorem 6.3 we show that the m'th randomized iterate of a weak one-way function along with the randomizing hash functions from a strong one-way function (for the right choice of m). Moreover, this holds also for the derandomized version of the randomized iterate (Theorem 6.7), giving an almost linear construction. Our construction is arguably simpler and has the following advantages:

1. While the [GIL+90] construction works only for *known* regular weak one-way functions, our amplification works for any regular weak one-way function (whether its image size is efficiently computable or not).

2. The input length of the resulting strong one-way function is $\Theta(n \log n)$ regardless of the required security. Thus, for some range of the parameters our solution is better than that of [GIL+90] (although it is worse than [GIL+90] for other ranges).

As in the case of pseudorandom generators discussed above, our method yields a construction with input length $\Theta(n)$ if bounded-space generators with better parameters become available.

The Idea. At the basis of all hardness amplification lies the fact that for any inverting algorithm, a weak one-way function has a set that the algorithm fails upon (hereafter, the *failing-set* of this algorithm). It follows that large enough number of randomly chosen inputs are bound to hit every such failing-set and thus to fail every algorithm. Taking independent random samples (i.e., $f'(x_1, \ldots, x_m) = (f(x_1), \ldots, f(x_m))$) works well ([Yao82]), but with the price of increasing the input length. An alternative approach (a variant of which was used by [GIL+90]) would be to use randomized iterations (i.e., to consider the function $f^m(x, \overline{h})$). This also amounts to applying f to m random inputs and therefore bound to hit every failing set. One obstacle is that when given $f^m(x, \overline{h})$, an adversary may invert f^m to a different input $(x', \overline{h}')$ (with different and carefully chosen hash functions) such that the computation of $f^m(x', \overline{h}')$ avoids applying f to a relevant failing set. To overcome this, the hash functions $\overline{h}$ are also given as part of the output (i.e., we consider the function $g(x, \overline{h}) = (f^m(x, \overline{h}), \overline{h})$), forcing an inverter to invert to the same hash functions. Using our core technique (from the pseudorandom generators section) we show that the hardness of inverting f^m is maintained even when $\overline{h}$ is known.

At a first glance, the aforementioned approach does not help in decreasing the input blowup, since the description of $\overline{h}$ is long. Indeed, choosing fully independent randomizing hash functions requires an input as long as that of Yao's solution (an input of length $\Theta(n \cdot \omega(\log(n))/\varepsilon)$). What makes this approach appealing, is the derandomization of the hash functions using space-bounded generators, which reduces the input length to only $\Theta(n \log n)$. We mention that since the hardness of f^m stems from the fact that a random input hits with high probability any failing-set, it is required that this is also the case for the derandomized f^m (and not only that the derandomized function maintains low collision probability as in the pseudorandom generator case). Fortunately, the derandomization using bounded-space generators also guarantees this property.

We mention that there have been several attempts to formulate such a construction, using all of the aforementioned tools. Goldreich et al. [GIL+90] did actually consider following the GKL methodology, but chose a different (though related) approach. Phillips [Phi92] gives a solution with input length $\Theta(n \log n)$ using bounded-space generators, but only for the simple case of permutations (where [GIL+90] has better parameters). Di Crescenzo and Impagliazzo [DI99] give a

solution for regular functions, but only in a model where public randomness is available (in the mold of [HL92]). Their solution is based on pairwise-independent hash functions that serve as the public randomness. We are able to combine all these ingredients into one general result, perhaps due to our simplified proof.

1.3.2 Additional issues

On non-length-preserving functions. This work focuses on length-preserving one-way functions. We also demonstrate how our proofs may be generalized, with no penalty in the tightness of the security, to use non-length preserving functions.[8] This generalization requires the use of a construction of a family of *almost* pairwise-independent hash functions (see Section 2.6.1 and Section 3.4.1).

The results in the public randomness model. Similarly to previous works, our results also give linear reductions in the public randomness model. This model (introduced by Herzberg and Luby [HL92]) allows the use of public random coins that are not regarded a part of the input. Our results, however, introduce significant savings in the amount of public randomness that is necessary.

1.4 Organization

Section 2 includes the formal definitions and notations used throughout this work. In Section 3 we present our construction of pseudorandom generators from regular one-way functions. Section 4 presents the construction based on exponentially-hard one-way functions and in particular proves a lemma regarding the hardness of inverting the randomized iterate of a general one-way function (Lemma 4.1). In Section 5 we present our improvement to the HILL construction of pseudorandom generators from any one-way function. Finally, in Section 6 we present our hardness amplification of regular one-way functions.

2 Preliminaries

2.1 Notations

We use capital letters for random variables and matrices, standard letters for values and calligraphic for sets. Given two equal length strings x and y, we denote by $\langle x, y \rangle_2$ their inner product modulus two. A set $\mathcal{L} \subseteq \mathcal{S}$ is of *density* (at least) δ with respect to $\mathcal{S}$, if $|\mathcal{L}| \geq |\mathcal{S}| \cdot \delta$. For $f \colon \mathcal{S} \mapsto \{0,1\}^*$, we let $\mathrm{Dom}(f) = \mathcal{S}$ and for $\mathcal{L} \subseteq \mathcal{S}$ let $\mathrm{Im}(f(\mathcal{L})) = \{f(x) \colon x \in \mathcal{L}\}$ (for $\mathcal{L} = \mathcal{S}$, we simply write $\mathrm{Im}(f)$). For $y \in \mathrm{Im}(f)$, we denote the preimages of y under f by $f^{-1}(y)$. The *degeneracy* of f on y is defined by $\mathrm{D}_f(y) \stackrel{\text{def}}{=} \left\lceil \log \left| f^{-1}(y) \right| \right\rceil$.

Through the paper we let n be the security parameter, and when its value is clear from to context, we sometimes omit it from the notation (e.g., we use ℓ instead of $\ell(n)$). A function $\mu : \mathbb{N} \mapsto [0,1]$ is *negligible*, if for every $p \in \mathrm{poly}$ it holds that $\mu(n) < 1/p(n)$ for large enough n. We denote by $f : \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$, the ensemble of functions $\left\{ f_n : \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)} \right\}_{n \in \mathbb{N}}$.

[8] Some alternative techniques for converting arbitrary one-way functions to length-preserving ones (e.g., padding), incur serious deterioration in the security of the resulting one-way functions.

2.2 Distributions and Entropy

We adopt the convention that when the same random variable appears multiple times in an expression, all occurrences refer to the same instantiation. For example, $\Pr[X = X]$ is 1. The *support* of a random variable X is $\mathrm{Supp}(X) \stackrel{\text{def}}{=} \{x : \Pr[X = x] > 0\}$. Let X be a random variable taking values in a finite set $\mathcal{U}$, then we write $x \leftarrow X$ to indicate that x is selected according to X. If $\mathcal{S}$ is a subset of $\mathcal{U}$, then $x \leftarrow \mathcal{S}$ means that x is selected according to the uniform distribution on $\mathcal{S}$. We write U_n to denote the random variable distributed uniformly over $\{0,1\}^n$. Given a function $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, we denote by $f(U_n)$ the distribution over $\{0,1\}^{\ell(n)}$ induced by f operating on the uniform distribution. By a distribution ensemble over $\{\mathcal{S}_n\}_{n\in\mathbb{N}}$, we mean a series $\{D_n\}_{n\in\mathbb{N}}$, where each D_n is a distribution over $\mathcal{S}_n$.
Let D be a distribution over some finite domain X, we use the following measures of entropy:

- The Shannon entropy of D, denoted $\mathrm{H}(D)$, is defined as $\sum_{x\in X} D(x) \cdot \log \frac{1}{D(x)}$.
- The collision probability of D, denoted $\mathrm{CP}(D)$, is defined as $\sum_{x\in X} D(x)^2$.
- The min entropy of D, denoted $\mathrm{H}_\infty(D)$, is defined as $\min_{x\in X} \log \frac{1}{D(x)}$.

Two distributions X and Y over $\mathcal{U}$ are ε close, if $\max_{\mathcal{S}\subseteq\mathcal{U}} |\Pr_{w\leftarrow X}(\mathcal{S}) - \Pr_{w\leftarrow Y}(\mathcal{S})| \leq \varepsilon$. We define the distinguishing advantage of an algorithm D between two distribution ensembles $\{X_n\}$ and $\{Y_n\}$ by

$$\Delta^D(X_n, Y_n) = |\Pr[D(1^n, x) = 1] - \Pr[D(1^n, y) = 1]|,$$

where the probabilities are taken over $x \leftarrow X_n$ and $y \leftarrow Y_n$, and the randomness of D.

2.3 Family of Pairwise-Independent Hash Functions

Definition 2.1 (family of pairwise-independent hash functions)**.** *An ensemble of function families* $\left\{\mathcal{H}_n = \left\{h\colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\right\}\right\}_{n\in\mathbb{N}}$ *is an* efficient family of pairwise-independent hash functions*, if* $\mathcal{H}_n$ *is polynomial-time samplable and computable,*[9] *and for any* $n \in \mathbb{N}$ *and* $x \neq x' \in \{0,1\}^n$ *it holds that* $(h(x), h(x'))_{h\leftarrow\mathcal{H}_n}$ *is uniformly distributed over* $\{0,1\}^{2\ell(n)}$*.*

In the following when n is clear from the context, we let $\mathcal{H} = \mathcal{H}_n$. There are various constructions of efficient families of pairwise-independent hash functions for any polynomial-time computable function $\ell(n) \in \mathrm{poly}(n)$, whose description length is linear in $n + \ell(n)$ (e.g., [CW79]). In this work we also make use of the special case in which $\ell(n) = n$ (the hash is length preserving). In such a case, $\mathcal{H}$ is called an efficient family of pairwise-independent length-preserving hash functions.

In some cases we cannot afford to use hash functions whose description length is linear in the input size, but can only afford a description that is linear in the output size. In such cases we use the following relaxation of pairwise-independent hash functions:

Definition 2.2 (family of almost pairwise-independent hash functions)**.** *An ensemble of function families* $\mathcal{H}_n = \left\{h\colon \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}\right\}_{n\in\mathbb{N}}$ *is an* efficient family of $\varepsilon(n)$-almost pairwise-independent hash functions*, if* $\mathcal{H}_n$ *is polynomial-time samplable and computable, and for all* $n \in \mathbb{N}$ *and* $x \neq x' \in \{0,1\}^n$ *it holds that* $(h(x), h(x'))_{h\leftarrow\mathcal{H}_n}$ *is* $\varepsilon(n)$*-close to the uniform distribution over* $\{0,1\}^{2\ell(n)}$*.*

[9] There exist two polynomial-time algorithm *Sam* and *Eval* such that the following hold for any $n \in \mathbb{N}$: $Sam(1^n)$ is uniform in $\mathcal{H}_n$, and $Eval(h, x) = h(x)$, for any $h \in \mathcal{H}_n$ and $x \in \{0,1\}^n$ (where both properties hold with respect to some fix description of the elements of $\mathcal{H}_n$).

Due to [CW79, WC81] and [NN93], for any polynomial-time computable functions ε and ℓ, where $\ell(n)$ is an integer function bounded by $\text{poly}(n)$, there exists such a family whose description length is $\Theta(\log(n) + \ell(n) - \log(\varepsilon(n)))$.

2.4 Randomness Extractors

Randomness extractors, introduced by Nisan and Zuckerman [NZ96], are an information theoretic tool for obtaining true randomness from a "weak" source of randomness. In this work, extractors are used in a computational setting to extract pseudorandomness from an imperfect source.

Definition 2.3 (strong extractors)**.** *A polynomial-time computable function* $\text{Ext} : \{0,1\}^{q(n)} \times \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ *is an (explicit)* $(k(n), \varepsilon(n))$-strong extractor*, if for every distribution* X *over* $\{0,1\}^n$ *with* $\text{H}_\infty(X) \geq k(n)$*, the distribution* $(\text{Ext}(U_{q(n)}, X), U_{q(n)})$ *is* $\varepsilon(n)$*-close to* $(U_{\ell(n)}, U_{q(n)})$.

2.5 Bounded-Space Generators

Bounded-space generators are (pseudorandom) generators against bounded-space adversaries. Such generators plays a central role in derandomization tasks. We are interested in generator for the following type of adversaries:

Definition 2.4 (bounded-width layered branching program - LBP)**.** *An* (s, m, v)-LBP M *is a finite directed acyclic graph whose nodes are partitioned into* $m+1$ *layers indexed by* $\{1, \ldots, m+1\}$*. The first layer has a single node (the source), the last layer has two nodes (sinks) labeled with 0 and 1, and each of the intermediate layers has up to* 2^s *nodes. Each node in the* $i \in [m]$ *layer has exactly* 2^v *outgoing labeled edges to the* $(i+1)$ *layer, one for every possible string* $z \in \{0,1\}^v$*.*

For a sequence $\overline{z} \in \{0,1\}^{mv}$*, we let* $M(\overline{z})$ *(the output of* M *on input* $\overline{z}$*) be the label reached at the end of the flowing* m*-step walk: the walk starts at the source node of the first layer, and at each step advances to the* $(i+1)$*'th layer along the edge labeled by* $\overline{z}_i$*.*

An alternative (and somewhat more intuitive) description to the above, associates labels with the graph's nodes (rather than with its edges). Upon "reading" the input $\overline{z}_i$, the program uses an *arbitrary* computation, which depends only on the current node label and $\overline{z}_i$, and advances to a node in the $i+1$ layer.

Definition 2.5. *A generator* $\text{BSG} : \{0,1\}^n \mapsto \{0,1\}^{mv}$ *is said to* ε-fool *an* LBP M *if*

$$|[M_x(U_{mv}) = 1] - \Pr[M(\text{BSG}(U_n)) = 0]| < \varepsilon.$$

The following theorem immediately follows from [INW94, Theorem 2].

Theorem 2.6 ([Nis92, INW94])**.** *Let* $s(n), m(n), v(n) \in \mathbb{N}$ *and* $\varepsilon(n) \in (0,1)$ *be polynomial-time computable functions. Then there exist a polynomial-time computable function* $q(n) \in \Theta(v(n) + (s(n) + \log(m(n)/\varepsilon(n))) \cdot \log m(n))$ *and a generator* $\text{BSG} : \{0,1\}^{q(n)} \mapsto \{0,1\}^{m(n)\cdot v(n)}$ *that runs in time* $\text{poly}(s(n), m(n), v(n), \log(1/\varepsilon(n)))$*, and* $\varepsilon(n)$*-fools every* $(s(n), m(n), v(n))$-LBP.

2.6 One-Way Functions

Definition 2.7 (one-way functions)**.** *A function $f : \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$ is $(T(n), \varepsilon(n))$-*one-way*, if f is polynomial-time computable and*

$$\Pr[A(1^n, f(U_n)) \in f^{-1}(f(U_n))] < \varepsilon(n)$$

for every algorithm A of running time $T(n)$ and large enough n.[10] *A* one-way permutation *is a one-way function that is a permutation over $\{0,1\}^{d(n)}$ for every $n \in \mathbb{N}$. Where f is* regular*, if there exists an integer function α such that*

$$\left|f^{-1}(f(x))\right| = \alpha(n)$$

for every $n \in \mathbb{N}$ and $x \in \{0,1\}^{d(n)}$. In the special case that α is polynomial-time computable, we say that f is known *regular.*[11]

In the case that $\varepsilon(n) = 1/T(n)$, we simply write that f is $T(n)$-one-way. f is one-way, if it is $T(n)$-one-way for every $T \in$ poly, where f is exponentially hard (one-way), if it is 2^{cn}-one-way for some constant $c > 0$. Finally, if f is $(T(n) > n^{O(1)}, 1 - \varepsilon(n))$-one-way, it is customary to call it an $\varepsilon(n)$-weak one-way function.

Save but in Sections 2.6.1 and 6, it will be the case that $d(n) = n$. The following observation tells us that in many cases (and in particular, in the cases mentioned above), it is possible to transform any one-way function (possibly only defined on a strict subset of the input lengths) into a one-way function with similar security, defined over all input lengths.

Proposition 2.8. *Assume there exist a $(T(n), \varepsilon(n))$-one-way function $f\colon \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$, where $d(n)$ is polynomial-time computable. Then there exists a $(T(r(n))/n^{O(1)}), \varepsilon(r(n)))$-one-way function $f'\colon \{0,1\}^n \mapsto \{0,1\}^{\ell(r(n))+n-r(n)}$, where $r(n) \stackrel{\text{def}}{=} \max\{n' \in [n] \colon d(n') \leq n\}$.*

Note that f' is length preserving iff f is, and that for slowly increasing d (i.e., $d(n+1)/d(n) \in O(1)$), the security of f' is similar to that of f.

Proof. Define $f'\colon \{0,1\}^n \mapsto \{0,1\}^{\ell(r(n))+n-r(n)}$ as

$$f'(x) = (f(x_{1,\ldots,r(|x|)}), x_{r(|x|)+1,\ldots,n}).$$

Clearly f' can be implemented to run in polynomial time, where the security of f' follows via the following argument: assume there exists an inverter A' that breaks the security of f', and consider the following inverter A for breaking the security of f: on input 1^n and $y \in \{0,1\}^{\ell(n)}$, apply A' on $(y, U_{i-d(n)})$ for every i such that $\min\{n, d(n)\} \leq i < \max\{n+1, d(n+1)\}$. If one of these applications succeeds and outputs a preimage $x \in \{0,1\}^i$, return $x_{d(n)}$ as the answer.

Let $\mathcal{I}$ be the infinite set of security parameters on which A' violates the security of f'. It follows that A inverts f with probability greater than $\varepsilon(n)$ on every $n \in r(\mathcal{I}) \stackrel{\text{def}}{=} \{r(i) \colon i \in \mathcal{I}\}$. Since $d(n) \leq \text{poly}(n)$ (as otherwise f cannot be computed in polynomial time), it follows that A runs in time $T(n)$ and that the set $r(\mathcal{I})$ is infinite, in contradiction to the one wayness of f. □

[10] We typically omit the security parameter (i.e., 1^n) from the adversary's parameters list.

[11] In this work we do not require such property, and our results hold for functions with unknown regularity. Thus, when we say regular functions we actually mean unknown-regular functions.

2.6.1 Length preserving one-way functions

In the following we prove the "folklore" fact that a one-way function can be assumed without loss of generality to be length preserving. In the case where the function is length decreasing, one can generate a length preserving one-way function simply by padding the output with extra zeros. In the case that it is length increasing, however, one needs to be more careful in order for the input length to remain of the same order.

Lemma 2.9. *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a $(T = T(n), \varepsilon = \varepsilon(n))$-one-way function and let $\mathcal{H}$ be an efficient family of 2^{-2n}-almost pairwise-independent hash functions from $\{0,1\}^{\ell(n)}$ to $\{0,1\}^{2n}$. We define g as*

$$g(x_a, x_b, h) = (h(f(x_a)), h),$$

where $x_a, x_b \in \{0,1\}^n$ and $h \in \mathcal{H}$. Then g is a length-preserving $(T - n^{O(1)}, \varepsilon + 2^{-n+1})$-one-way function.[12]

Proof. The function g is length preserving as both input and output are of length $2n$ plus the description of $h \in \mathcal{H}$. Let A be an algorithm that runs in time $T_A = T_A(n)$ and inverts g with probability $\varepsilon_A = \varepsilon_A(n)$. Note that x_b is a dummy input (used just for padding) so the success of A is taken over (x_a, h) and A's randomness. Define M^A as follows:

Algorithm 2.10. *(M^A)*

Input: $y \in f(\{0,1\}^n)$.

Operation:

1. *Choose a uniformly random $h \in \mathcal{H}$.*
2. *Apply $A(h(y), h)$ to get an output (x_a, x_b, h).*
3. *Output x_a.*

Clearly, the running time of M^A is (at most) $T_A + n^{O(1)}$. Algorithm M^A is sure to succeed on any choice of (y, h) for which the following hold: A succeeds on $(h(y), h)$ and there exists no $y' \neq y \in f(\{0,1\}^n)$ such that $h(y') = h(y)$ (h does not introduce any collision to y). Let $\mathcal{S} \stackrel{\text{def}}{=} \{(y,h) \in \{0,1\}^n \times \mathcal{H} \ : \ \exists y' \neq y \in f(\{0,1\}^n) \ : \ h(y') = h(y)\}$. Thus,

$$\begin{aligned}\Pr[M^A(f(U_n)) \in f^{-1}(f(U_n))] &\geq \Pr[A(g(U_n, H)) \in g^{-1}(g(U_n, H)) \wedge g(U_n, H) \notin \mathcal{S}] \\ &\geq \Pr[A(g(U_n, H)) \in g^{-1}(g(U_n, H))] - \Pr[g(U_n, H) \in \mathcal{S}],\end{aligned}$$

where H is a random variable uniformly distributed over $\mathcal{H}$. The almost pairwise independence of $\mathcal{H}$ yields that for every $y \neq y' \in f(\{0,1\}^n)$, it holds that $\Pr[H(y') = H(y)] \leq 2^{-2n+1}$. Since $|f(U_n)| \leq 2^n$, a union bound implies that $\Pr[\exists y' \neq y \in f(\{0,1\}^n) \ : \ H(y') = H(y)] \leq 2^{-n+1}$ for every $y \in f(U_n)$. Thus, an averaging argument yields that $\Pr[(f(U_n), H) \in \mathcal{S}] \leq 2^{-n+1}$. Putting it all together, we get that $\Pr[M^A(f(U_n)) \in f^{-1}(f(U_n))] \geq \varepsilon_A - 2^{-n+1}$. □

[12] Since a member of $\mathcal{H}$ can be describes using $\Theta(n)$ bits, the input length of g (and therefore also its output length) is in the same order of that of f.

2.7 Hardcore Predicates and Functions

Hard-core predicates/functions have a major role in the construction of one-way functions based pseudorandom generators.

Definition 2.11 (hardcore functions)**.** *We call* $hc : \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$ *a* $(T(n), \varepsilon(n))$-hardcore function *of* $f : \{0,1\}^{d(n)} \mapsto \{0,1\}^*$ *over* $\mathcal{S}_n \subseteq \{0,1\}^{d(n)}$*, if*

$$\Delta^{D^{f_n, hc_n}}\left((d(n), f(x), hc(x))_{x \leftarrow \mathcal{S}_n}, (d(n), f(x), U_{\ell(n)})_{x \leftarrow \mathcal{S}_n}\right) \leq \varepsilon(n),$$

for every oracle-aided algorithm D *of running time* $T(n)$ *and large enough* n *(where* f_n *and* hc_n *are the restrictions of* f *and* hc *to inputs of length* $d(n)$*).*[13]

We use the following conventions: in case $\mathcal{S}_n = \{0,1\}^{d(n)}$, we omit it from the above notation. In the case that $\varepsilon = 1/T$, we simply say that hc is a T-hardcore function. hc is simply a hardcore function, if it is $T(n)$-hardcore function for every $T \in$ poly. If hc is a predicate (i.e., $\ell(n) = 1$), it is called a hardcore predicate of f.[14] Finally, it is common to call the value $hc(x)$, the "hardcore-bits" of $f(x)$.

It is easy to see that a $(T(n), \varepsilon(n))$-hardcore function of f over $\mathcal{S}_n \subseteq \{0,1\}^{d(n)}$ with density $\delta(n)$ (i.e., $|\mathcal{S}_n| \geq \delta(n) \cdot 2^{d(n)}$), is a $(T(n), \varepsilon(n) + (1 - \delta(n))/2)$-hardcore function of f over $\{0,1\}^{d(n)}$.

We use the Goldreich-Levin hardcore functions, and in particular make use of the following theorem whose proof immediately follows from [GL89, Corollary 1].

Theorem 2.12 ([GL89])**.** *There exists a polynomial-time computable function* $gl : \{0,1\}^{3v} \mapsto \{0,1\}^v$ *such that the following holds: let* $d(n), m_f(n), m_g(n) \leq \text{poly}(n)$ *be integer functions, let* $f : \{0,1\}^{d(n)} \mapsto \{0,1\}^{m_f(n)}$*,* $g : \{0,1\}^{d(n)} \mapsto \{0,1\}^{m_g(n)}$ *and let* $\mathcal{S}_n \subseteq \{0,1\}^{d(n)}$*. Assume that*

$$\Pr_{x \leftarrow \mathcal{S}_n}[A^{f_n, g_n}(1^n, 1^{d(n)}, f(x)) = g(x)] \leq \varepsilon = \varepsilon(n)$$

for any oracle-aided algorithm A *of running time* $T = T(n)$ *and large enough* n*, where (*f_n *and* g_n *are the restrictions of* f *and* g *to inputs of length* $d(n)$*). Then for any integer function* $\ell(n) \in [m_g(n)]$*, the function* $hc_\ell : \{0,1\}^{d(n)} \times \{0,1\}^{2m_g(n)} \mapsto \{0,1\}^{\ell(n)}$*, defined as* $hc_\ell(x, r) = gl(g(x), r)_{1,\ldots,\ell(n)}$*, is a* $(T \cdot (\varepsilon/n)^{O(1)}, O(2^{\ell(n)}\varepsilon(n)))$*-hardcore function of* $f'(x, r) = (f(x), r)$ *over* $\mathcal{S}'_n = \mathcal{S}_n \times \{0,1\}^{2m_g(n)}$*.*

The following observation is immediate:

Corollary 2.13. *Assuming that* $T(n) = 1/\varepsilon(n) > n^{O(1)}$*, then for any integer function* $\ell(n) \in [\lfloor c \cdot \log T(n) \rfloor]$*, where* $c > 0$ *is a universal constant, the function* hc_ℓ *is a* $T(n)^{\Omega(1)}$*-hardcore function of* f' *over* $\mathcal{S}'_n$*.*

[13] By giving D oracle access to f and hc, we make the above definition applicable to the (not uniformly computable) hardcore functions that arise from the randomized iterate constructions. Note that if both f and hc are polynomial-time computable, then the definition above agrees with the common definition of hardcore functions.

[14] For hardcore predicates, it is more common (and somewhat more convenient) to measure the advantage of a possible "predictor" in guessing $hc(x)$ given $f(x)$, rather than the distinguishing gap as above. Yet, we preferred the above definition to have the same terminology for (hardcore) predicates and functions.

2.8 A Uniform Extraction Lemma

The following lemma is a generalization of the (uniform version) of Yao's XOR lemma. Given t independent $(T, (1-\varepsilon)/2)$-hardcore bits, we would like to extract approximately εt pseudorandom bits out of them. The version we present here generalizes [HILL99, Lemma 6.5]). In particular, the original lemma required the predicate to have a hardcore set (i.e., a subset of inputs on which the predicate is very hard), where in the following lemma this property is no longer required. In addition, the original lemma was tailored for a specific function/predicate pair, where the following lemma suits any such pair. Finally, the original lemma is stated using an efficient family of pairwise-independent hash functions, while the following lemma is stated using any explicit randomness extractor. The lemma is proven using Holenstein's "uniform hardcore lemma" [Hol05].[15]

Lemma 2.14. *Let $\ell(n), d(n), t(n), r(n)$ and $q(n)$ be integer functions bounded by $\mathrm{poly}(n)$ where $d(n) \geq n$, let $\varepsilon(n) \in [0,1] > 1/\mathrm{poly}(n)$ be polynomial-time computable function and let $\alpha \in (0,1)$. Let $\mathrm{Ext} : \{0,1\}^{t(n)} \times \{0,1\}^{q(n)} \mapsto \{0,1\}^{e(n)}$ be an $(\alpha \cdot t(n) \cdot \varepsilon(n), \varepsilon_{\mathrm{Ext}}(n))$-strong-extractor, let $f : \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$ and $b\colon \{0,1\}^{d(n)} \mapsto \{0,1\}$. We define the function $hc\colon \{0,1\}^{t(n)\times d(n)} \times \{0,1\}^{q(n)} \mapsto \{0,1\}^{r(n)}$ as*

$$hc(x_1, \ldots, x_t, s) = \mathrm{Ext}(s, b(x_1), \ldots, b(x_t)).$$

Assume that b is a $(T(n), (1-\varepsilon)/2)$-hardcore predicate of f over $\{0,1\}^{d(n)}$, then hc is a $(T \cdot (\gamma(n)/n)^{O(1)}, \varepsilon_{\mathrm{Ext}}(n) + \rho(n) + \gamma(n))$-hardcore function of $f'(x_1, \ldots, x_t, s) = (f(x_1), \ldots, f(x_t), s)$ over $\{0,1\}^{d'(n)=t(n)d(n)+q(n)}$, where $\gamma(n) > 2^{-n/4}$ is any polynomial-time computable function, and $\rho(n) = 2^{-\Omega(t(n)\cdot\varepsilon(n)^2)}$.

Proof. Assume towards a contradiction that there exists an oracle-aided algorithm D that runs in time $T_D(n) = T(n) \cdot (\gamma(n)/n)^{O(1)}$ and

$$\begin{aligned} \varepsilon_D(n) &= \Delta^{D^{f'_n, hc_n}}\left((1^{d'(n)}, f(x), hc(x))_{x\leftarrow\{0,1\}^{d'(n)}}, (1^{d'(n)}, f(x), U_{r(n)})_{x\leftarrow\{0,1\}^{d'(n)}}\right) \\ &> \varepsilon_{\mathrm{Ext}}(n) + \rho(n) + \gamma(n), \end{aligned} \tag{1}$$

for infinitely many n's. Fix $n \in \mathbb{N}$ such that the above equation holds, and for a fixed set $\mathcal{S} \subseteq \{0,1\}^d$ with $|\mathcal{S}| \geq \varepsilon \cdot 2^d$, define the following, not necessarily efficiently computable, randomized predicate $Q : \{0,1\}^n \mapsto \{0,1\}$:

$$Q(x) = \begin{cases} U_1 & x \in \mathcal{S}, \\ b(x) & \text{otherwise.} \end{cases} \tag{2}$$

Now for $i \in \{0, \ldots, t\}$ define the random variable X^i as

$$X^i = (f(U_n^1), \ldots, f(U_n^t), U_q, \mathrm{Ext}(U_q, b(U_n^1), \ldots, b(U_n^i), Q(U_n^{i+1}), \ldots, Q(U_n^t)).$$

For the right constant in the definition of ρ, a Chernoff bound yields that the distribution $(Q(U_n^1), \ldots, Q(U_n^t)) \mid (f(U_n^1), \ldots, f(U_n^t))$ is ρ-close to a distribution with min-entropy $\alpha \cdot t \cdot \varepsilon$. Hence, the statistical distance between X^0 and $(f(U_n^1), \ldots, f(U_n^1), U_q, U_r)$ is bounded by $\varepsilon_{\mathrm{Ext}} + \rho$. Since $X^t = (f(U_d), hc(U_d))$, it follows that

$$\left| \Pr_{w\leftarrow X^0}[D^{f'_n, hc_n}(1^{d'}, w) = 1] - \Pr_{w\leftarrow X^t}[D^{f'_n, hc_n}(1^{d'}, w) = 1] \right| > \varepsilon_D - \varepsilon_{\mathrm{Ext}} - \rho = \gamma.$$

[15] Independently of this work, [Hol06b, Theorem 7.3] presents a similar variant of Lemma 2.14.

A standard hybrid argument yields the existence of an index $j \in \{0,\ldots,t-1\}$ such that D distinguishes between X^j and X^{j+1} with advantage γ/t. Since X^j and X^{j+1} are identical conditioned that $U_n^{j+1} \notin \mathcal{S}$, it follows that D achieves this advantage when conditioning that $U_n^{j+1} \in \mathcal{S}$. Consider the oracle-aided algorithm B that given oracle access to f_n, b_n and $\chi_{\mathcal{S}}$ — the characteristic function of $\mathcal{S}$, distinguishes between $(1^d, f(x), b(x))_{x\leftarrow\mathcal{S}}$ and $(1^d, f(x), U)_{x\leftarrow\mathcal{S}}$ as follows: B first uses D and the oracles to find t, q, d' and $j \in \{0,\ldots,t-1\}$ such that

$$\Pr_{w\leftarrow X^j}[D^{f'_n,hc_n}(1^{d'},w)=1] - \Pr_{w\leftarrow X^{j+1}}[D^{f'_n,hc_n}(1^{d'},w)=1] > \gamma/2t.$$

Note that using $n^{O(1)}/\gamma$ calls to D, f_n, b_n and $\chi_{\mathcal{S}}$, such index can found correctly with probability $1-2^{-n}$ by trying all $(\mathrm{poly}(n))$ possible values for the parameters, and use f_n and b_n to answer calls to f'_n and hc_n.[16] On input $(1^d, f(x), b)$, B samples random value $w = (f(x^1),\ldots,f(x^t),b_1,\ldots,b^t)$ from X^{j+1}, replaces $f(x^{j+1})$ and b^{j+1} with $f(x)$ and b respectively, and returns $D^{f'_n,hc_n}(1^{d'},w)$. Hence, B runs in time $T_B = T_D \cdot n^{O(1)}/\gamma$, and

$$\Pr_{x\leftarrow\mathcal{S}}[B^{f_n,b_n,\chi_{\mathcal{S}}}(1^d,f(x),b(x))=1] - \Pr_{x\leftarrow\mathcal{S}}[B^{f_n,b_n,\chi_{\mathcal{S}}}(1^d,f(x),U)=1] > \gamma/2t - 2^{-n} > \gamma/3t.$$

Using a standard reduction from distinguishing to predicting, we obtain an oracle-aided predictor P, running in time $T_B + O(1)$, such that

$$\Pr_{x\leftarrow\mathcal{S}}[P^{f_n,b_n,\chi_{\mathcal{S}}}(1^d,f(x)) = b(x)] > \frac{1}{2} + \gamma/3t \tag{3}$$

Specifically, on input $(1^d, y)$, P generates a random bit $c \leftarrow \{0,1\}$, runs $B^{f_n,b_n,\chi_{\mathcal{S}}}(1^d,y,c)$, outputs c if B outputs 1, and outputs $1-c$ if B outputs 0. We complete the proof using the following proposition that immediately follows from [Hol06b, Theorem 6.8].

Proposition 2.15. *(Uniform Hardcore Lemma, [Hol05, Hol06a, Hol06b]) Let $\delta(n) \in [0,1] > 1/\mathrm{poly}(n)$ and $\gamma(n) \in [0,1] > 2^{-n/3}$ be polynomial-time computable functions. Let $d(n) \geq n, \ell(n) \leq \mathrm{poly}(n)$, $f\colon \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$ and $b\colon \{0,1\}^{d(n)} \mapsto \{0,1\}$. Assume that*

$$\Pr[M^{f_n,b_n}(1^{d(n)}, f(U_{d(n)})) = b(U_{d(n)})] \leq 1 - \delta(n)/2$$

for every oracle-aided predictor M running in time $T(n)$ and large enough n. Then for every oracle-aided predictor P running in time $T(n)\cdot(\gamma(n)/n)^{O(1)}$, for large enough n there exists a set $\mathcal{S}_n \subseteq \{0,1\}^{d(n)}$ with $|\mathcal{S}_n| \geq \delta(n)\cdot 2^{d(n)}$ such that

$$\Pr_{W\leftarrow\mathcal{S}_n}[P^{\chi_{\mathcal{S}_n},f_n,b_n}(1^{d(n)},f(W)) = b(W)] \leq (1+\gamma(n))/2,$$

provided that all the queries of P to $\chi_{\mathcal{S}}$ are computed independently of the input $f(W)$.[17]

[16] We assume without loss of generality such j always exists, as otherwise we can consider the negation of D.

[17] [Hol06b, Theorem 6.8] requires f and b to be polynomial-time computable and their hardness to hold with respect to any (large enough) input length. In the case studied below, however, f and b are given as oracles (for the given input length) and they are only defined over (infinite) subset of all input lengths. In addition, [Hol06b, Thm 6.8] is only stated with respect to $\gamma > 1/\mathrm{poly}(n)$, where below we only ask $\gamma > 2^{-n/3}$. Nevertheless, the generalization presented here readily follows from the original proof.

Using Proposition 2.15, Eq(3) yields that there exists a predictor M that runs in time $T_B \cdot (\gamma/n)^{O(1)} = T_D \cdot (\gamma/n)^{O(1)} \leq T - \text{poly}(n)$ (for the right constant in the definition of T_D) and

$$\Pr[M^{f_n,b_n}(1^d, f(U_d)) = b(U_d)] > 1 - \varepsilon/2,$$

for infinitely many n's. Hence the algorithm that on input (y, b) output 1 if $M(y) = b$ and 0 otherwise, runs in time at most T and distinguishes (for infinitely many n's) between $(1^d, f(U_d), b(U_d))$ and $(1^d, f(U_d), U)$ with advantage greater than $(1-\varepsilon)/2$, in contradiction to the assumption about f and b. □

2.9 Pseudorandom Generators

Definition 2.16 (pseudorandom generators)**.** *An ensemble of distributions $D = \{D_n\}_{n\in\mathbb{N}}$ over $\{0,1\}^{\ell(n)}$ is $(T(n), \varepsilon(n))$-*pseudorandom*, if*

$$\Delta^D\left(D_n, U_{\ell(n)}\right) < \varepsilon(n)$$

for every algorithm D of running time $T(n)$ and large enough n.

*A a function $G : \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$ is a $(T(n), \varepsilon(n))$-*pseudorandom generator*, if it is length-increasing and polynomial-time computable, and $G(U_{d(n)})$ is $((T(n), \varepsilon(n))$-pseudorandom.*

In the case that $\varepsilon(n) = 1/T(n)$, we simply say that G is a $T(n)$-pseudorandom generator, where G is a pseudorandom generator, if it is $T(n)$-pseudorandom generator for any $T \in$ poly.

Similarly to the case of one-way functions, the following observation tells us that in many settings (and in particular, in all the concrete constructions presented in this paper), it is possible to transform any pseudorandom generator (possibly only defined on a strict subset of the input lengths) into a pseudorandom generator with similar security, defined over all input lengths.

Proposition 2.17. *Assume there exist a $(T(n), \varepsilon(n))$-pseudorandom generator $G\colon \{0,1\}^{d(n)} \mapsto \{0,1\}^{\ell(n)}$, where $d(n)$ and $\varepsilon(n)$ are polynomial-time computable. Then there exists a $(T(r(n)) \cdot (\varepsilon(r(n))/n)^{O(1)}), 2\varepsilon(r(n)))$-pseudorandom generator $G'\colon \{0,1\}^n \mapsto \{0,1\}^{\ell(r(n))+n-r(n)}$, where $r(n) \stackrel{\text{def}}{=} \max\{n' \in [n]\colon d(n') \leq n\}$.*

Note that for slowly increasing d (i.e., $d(n+1)/d(n) \in O(1)$), the security of G' is similar security to that of G, and G' stretches its input in essentially the same rate as G does.

Proof. The proof follows similar line to that of Proposition 2.8. Define $G'\colon \{0,1\}^n \mapsto \{0,1\}^{\ell(r(n))+n-r(n)}$ as

$$G'(x) = (G(x_{1,\ldots,r(|x|)}), x_{r(|x|)+1,\ldots,n}).$$

Clearly G' can be implemented to run in polynomial time, where the security of G' follows via the following argument: assume there exists a distinguisher D' that breaks the security of G', we consider the following distinguisher D for breaking the security of G: on input 1^n and $y \in \{0,1\}^{\ell(n)}$, search for $\min\{n, d(n)\} \leq i < \max\{n+1, d(n+1)\}$ such that D' distinguishes the output of G' on input of length i from random, with maximal advantage. Return $D'(1^n, (y, U_{i-d(n)}))$.

Let $\mathcal{I}$ be the infinite set of security parameters on which D' violates the hardness of G'. Setting D to evaluate the advantage per i using $O(d(n)/\varepsilon(n))$ samples, and assuming without loss of generality that $\varepsilon(n) > 2^{-d(n)}$, it follows that D distinguishes the output of G on input of length $d(n)$ from random with advantage $2\varepsilon(n) - 2^{-d(n)} > \varepsilon(n)$ on any $n \in r(\mathcal{I}) \stackrel{\text{def}}{=} \{r(i)\colon i \in \mathcal{I}\}$. Since $d(n) \leq \text{poly}(n)$, it follows that D runs in time $T(n)$ and that the set $r(\mathcal{I})$ is infinite, in contradiction to the pseudorandomness of G. □

2.10 The Security of Cryptographic Constructions

Typically the proof of security for cryptographic constructions is based on reductions. In this paradigm we use a presumably secure implementation of one primitive (or possibly several primitives) in order to implement a second primitive. The proof of security for the second primitive relies on the security assumption for the original one. More precisely, we prove that any efficient adversary that breaks the implementation of the second primitive can be used to efficiently break the original primitive. Note that the meaning of "breaking a primitive" and, furthermore, the definition of the *success probability* of an adversary in breaking the primitive, varies between different primitives. For example, in the case of one-way functions the success probability is the fraction of inputs on which the adversary manages to invert the function. Usually, there is a tradeoff between the running time of an adversary and its success probability (e.g., it may be possible to utterly break a primitive by enumerating all possibilities for the secret key). Therefore, both the running time and success probability of possible adversaries are relevant when analyzing the security of a primitive. A useful, combined parameter is the *time-success ratio* of an adversary which we define next.

Definition 2.18 (time-success ratio)**.** *Let P be a primitive and let A be an adversary running in time $T_A(n)$ and breaking P with probability $\varepsilon_A(n)$. The* time-success ratio *of A in breaking P is defined as $R(n) = \frac{T_A(n)}{\varepsilon_A(n)}$, where n is the security-parameter of the primitive.*

Note that in the above definition, the smaller the R the better A is in breaking P. A quantitative analysis of the security of a reduction is crucial for both theoretical and practical reasons. Given an implementation of primitive P using primitive Q along with a proof of security, let R_P be the security-ratio of a given adversary with respect to P and let R_Q be the security-ratio of the adversary that the proof of security yields. A natural way to measure the security of a reduction is by the relation between R_P and R_Q. Clearly, the smaller the R_Q comparing to R_P, the better the performance of the adversary the reduction yields when trying to break Q comparing to the performance of the adversary trying to break P.

The most desirable reductions is when $R_Q(n) \in n^{O(1)} \cdot O(R_P(O(n)))$. In such reductions, known as *linear-preserving reductions*, we are guaranteed that breaking the constructed primitive is essentially as hard as breaking the original one. Next we find the *polynomial-preserving reductions* when $R_Q \in n^{O(1)} \cdot O(R_P(O(n))^{O(1)})$. Note that a linear/polynomial-preserving reduction typically means that for the same level of security we can take the inputs of Q and P to be of the same length (up to a constant-ratio). The other side of the scale is when $R_Q \in n^{O(1)} \cdot O(R_P(n^{O(1)}))$. In such reductions, known as *weak-preserving reductions*, we are only guaranteed that breaking P is as hard as breaking Q for polynomially smaller security-parameter (e.g., polynomially smaller input length). For a more comprehensive discussion of the above issues the reader may refer to [Gol00, HL92]. This quantitative classification of security preserving reduction partly motivates our focus on the input-length as the main parameter that our reductions aim to improve. In particular, better space bounded generators would make our reduction polynomial-preserving rather than weak-preserving (see Section 2.5).

2.10.1 Black-box reductions

It is worth mentioning that all the reductions considered in this paper (e.g., from length regular to length preserving one-way functions, Lemma 2.9) are fully black box – the underlying function

and the possible adversary are treated as black boxes (i.e., as oracles). The only exception is the bounded-space generator of Theorem 2.6, whose proof inherently uses the internal structure of the "adversary" (i.e., the branching program).

3 Pseudorandom Generators from Regular One-Way Functions

The following discussion considers only length preserving regular one-way functions, where the extension to general regular one-way functions is described in Section 3.4.1.

3.1 Some Motivation and the Randomized Iterate

Recall that the BMY generator simply iterates the one-way permutation f on itself, and outputs a hardcore-bit of the intermediate step at each iteration. The crucial point is that since f is a permutation, the output of the function on U_n is also uniform in $\{0,1\}^n$. Hence, when applying f to the output, it is hard to invert this last application of f, and therefore hard to predict the new hardcore-bit (Yao [Yao82] shows that the unpredictability of bits implies pseudorandomness). Since the seed is essentially just an n bit string and the output is as long as the number of iterations, the generator stretches its input.

We want to duplicate this approach for general one-way functions, but unfortunately the situation changes drastically when the function f is not a permutation. After a single application of f, the output may be very far from uniform, and in fact may be concentrated on a very small and easy fraction of the inputs to f. Thus, reapplying f to this output gives no hardness guarantees at all. In an attempt to salvage the BMY framework, Goldreich et al. [GKL93] suggested to add a randomization step between every two applications of f, making the next input to f a truly random one. This modification, which we call the randomized iterate, lies at the core of our work and is defined next.

Definition 3.1 (the randomized iterate)**.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^n$, let $\mathcal{H}$ be an efficient family of pairwise-independent length-preserving hash functions over $\{0,1\}^n$, and let $m(n) \in \mathbb{N}$. For an integer function $2 \leq k(n) \leq m(n)+1$, $x \in \{0,1\}^n$ and $\overline{h} \in \mathcal{H}^m$, define the k'th* randomized iterate *$f^k : \{0,1\}^n \times \mathcal{H}^m \mapsto \{0,1\}^n$ recursively as*

$$f^k(x,\overline{h}) = f(\overline{h}_{k-1}(f^{k-1}(x,\overline{h}))),$$

where $f^1(x,\overline{h}) = f(x)$.

In the following we denote by H^m the random variable uniformly distributed over $\mathcal{H}^m$.

The application of the randomized iterate for pseudorandom generators is a bit tricky. On the one hand, such a randomization costs a large number of random bits, much larger than what can be compensated for by the hardcore-bits generated in each iteration. So in order for the output to actually be longer than the input, we also output the descriptions of the hash functions. But on the other hand, handing out the randomizing hash gives information on intermediate values such as $f^i(x,\overline{h})$, and thus f might no longer be hard to invert when applied to such an input. Somewhat surprisingly, the randomized iterate of a regular one-way function remains hard to invert even when the hash functions are known. This fact, which is central to the whole approach, was proved in [GKL93] when using a family of n-wise independent hash functions. As a first step, we give a simpler proof that extends to pairwise-independent hash functions as well.

3.2 The Last Randomized Iteration is Hard to Invert

In this section we formally state and prove the key observation mentioned above. After applying k randomized iterations of a regular one-way function f, it is hard to invert the last iteration, even if given access to all of the hash functions leading up to this point.

Lemma 3.2. *Let f, m, k and f^k be as in Definition 3.1. Assume that f is regular, then for $n \in \mathbb{N}$ and any algorithm A with*

$$\varepsilon_A = \Pr[A(f^k(U_n, H^m), H^m) = f^{k-1}(U_n, H^m)],$$

it holds that

$$\Pr[H^m_{k-1}(A(f(U_n), H^m)) \in f^{-1}(f(U_n))] \geq \varepsilon_A^2/k.$$

Assuming that f is a one-way function, we get the following corollary:

Corollary 3.3. *Let f, k, m and f^k be as in Definition 3.1. Assume that f is one-way and that $m(n) \leq \text{poly}(n)$, then the following hold:*

1. $\Pr[A(f^k(U_n, H^m), H^m) = f^{k-1}(U_n, H^m)] = \text{neg}(n)$ *for any polynomial-time algorithm A, and*
2. *The predicate $b^k \colon \{0,1\}^n \times \mathcal{H}^m \times \{0,1\}^{2n} \mapsto \{0,1\}$ defined as $b^k(x, \overline{h}, r) = gl(f^{k-1}(x, \overline{h}), r)_1$ is a hardcore predicate of $\widehat{f^k}(x, \overline{h}, r) = (f^k(x, \overline{h}), \overline{h}, r)$, where gl is the Goldreich-Levin hardcore function (see Theorem 2.12).*

Proof. (of Corollary 3.3) Assume there exists a polynomial-time algorithm A that violates (1), then by Lemma 3.2 $\Pr[H^m_1(A(f(U_n), H^m)) \in f^{-1}(f(U_n))]$ is non negligible. Hence, there exists a polynomial-time algorithm that inverts f with non negligible probability.

For the second part of the lemma, assume towards a contradiction that hc^k is not such a hardcore function of $\widehat{f^k}$. Theorem 2.12 yields that there exists a polynomial-time oracle-aided algorithm A for which

$$\Pr_{x \leftarrow \mathcal{S}_n}[A^{f^k_n, f^{k-1}_n}(f^k(U_n, H^m), H^m) = f^{k-1}(U_n, H^m)] > 1/\text{poly}(n),$$

for infinitely many n's (where f^k_n and f^{k-1}_n are the restrictions of f^k and f^{k-1} to inputs in $\{0,1\}^n \times \mathcal{H}^m$). Since f^k_n and f^{k-1}_n are efficiently computable given k, the polynomial-time algorithm that tries all values of $k \in [m]$ and runs the above A for each such choice, inverts the last iteration of f^k with non-negligible probability, in contradiction to (1). □

Proof. (of Lemma 3.2) Note that $A(f^k(U_n, H^m), H^m) = f^{k-1}(U_n, H^m)$ implies that $H^m_{k-1}(A(f(U_n), H^m)) \in f^{-1}(f^k(U_n, H^m))$. Since $f^k(U_n, H^m)$ and $f(U_n)$ are identically distributed, and thus the lemma would have followed trivially had H^m been *independent* of $f^k(U_n, H^m)$. Unfortunately this is clearly not the case for arbitrary function f. Rather, Lemma 3.4 states that for our purposes, the variables H^m and $f^k(U_n, H^m)$ are "close enough" to being independent in the case of regular functions. Details follow.

In the following we assume that A is deterministic, where the proof for randomized A would follow.[18] Let $\mathcal{S}_A \subset \mathrm{Im}(f^k) \times \mathcal{H}^m$ include the inputs on which A is successful upon. Namely,

$$\mathcal{S}_A = \left\{(y, \overline{h}) \in \mathrm{Im}(f^k) \times \mathcal{H}^m : \overline{h}_{k-1}(A(y, \overline{h})) = y\right\}$$

Since $\Pr[(f^k(U_n, H^m), H^m) \in \mathcal{S}_A] = \varepsilon_A$, the proof of Lemma 3.2 follows by the next lemma when plugging in $\mathcal{L} = \mathcal{S}_A$ and $\delta = \varepsilon_A$.

Lemma 3.4. *Let f, m, k, and f^k be as in Lemma 3.2. Assume that f is regular, then for every set $\mathcal{L} \subseteq \mathrm{Im}(f^k) \times \mathcal{H}^m$, if*

$$\Pr[(f^k(U_n, H^m), H^m) \in \mathcal{L}] \geq \delta,$$

then

$$\Pr[(f(U_n), H^m) \in \mathcal{L}] \geq \delta^2/k.$$

□

Proof. (of Lemma 3.4) The lemma essentially states that with respect to $\widehat{f^k}(x, \overline{h}) = (f^k(x, \overline{h}), \overline{h})$ (i.e, the function defined by concatenating the input hash functions to the output of f^k), any large subset of inputs induces a large subset of outputs. Thus, there is a fairly high probability of hitting this output set simply by sampling independent y and $\overline{h}$. Intuitively, if a large set of inputs induces a small set of outputs, then there must be many collisions in this set (a collision means that two different inputs lead to the same output). Indeed, to prove the lemma we start by showing that many collisions are impossible, or more precisely, by proving an upper bound on the collision probability of the function $(f^k(x, \overline{h}), \overline{h})$.

Claim 3.5.

$$\mathrm{CP}(f^k(U_n, H^m), H^m) \leq \frac{k}{|\mathcal{H}|^m \cdot |\mathrm{Im}(f)|}.$$

Proof. For every two inputs $(x_0, \overline{h_0})$ and $(x_1, \overline{h_1})$ to f^k, in order to have a collision we must first have that $\overline{h_0} = \overline{h_1}$, which happens with probability $\frac{1}{|\mathcal{H}|^m}$. Now, given that $\overline{h_0} = \overline{h_1} = \overline{h}$ (with a random $\overline{h} \in \mathcal{H}^m$), we require also that $f^k(x_0, \overline{h})$ equals $f^k(x_1, \overline{h})$. If $f(x_0) = f(x_1)$ (happens with probability $1/|\mathrm{Im}(f)|$) then a collision is assured. Otherwise, there must be an $i \in [k-1]$ for which $f^i(x_0, \overline{h}) \neq f^i(x_1, \overline{h})$, but $f^{i+1}(x_0, \overline{h}) = f^{i+1}(x_1, \overline{h})$. Since $f^i(x_0, \overline{h}) \neq f^i(x_1, \overline{h})$, due to the pairwise independence of h_i, the values $h_i(f^i(x_0, \overline{h}))$ and $h_i(f^i(x_1, \overline{h}))$ are uniformly random values in $\{0,1\}^n$, and thus $f(h_i(f^i(x_0, \overline{h}))) = f(h_i(f^i(x_1, \overline{h})))$ happens with probability $1/|\mathrm{Im}(f)|$. Altogether, $\mathrm{CP}(f^k(U_n, H^m), H^m) \leq \frac{1}{|\mathcal{H}|^m} \cdot \frac{k}{|\mathrm{Im}(f)|}$. □

[18] Let N be the number of different values for the coins used by A, let A_r be the instance of A whose random coins are fixed to r and let ε_{A_r} be the success probability of A_r. Assuming Lemma 3.2 for deterministic algorithms, it follows that

$$\begin{aligned}\Pr[H^m_{k-1}(A(f(U_n), H^m)) \in f^{-1}(f(U_n))] &= \frac{1}{N} \cdot \sum_r \Pr[H^m_{k-1}(A_r(f(U_n), H^m)) \in f^{-1}(f(U_n))] \\ &\geq \frac{1}{N} \cdot \sum_r \frac{\varepsilon^2_{A_r}}{k} = \frac{1}{Nk} \cdot \sum_r \varepsilon^2_{A_r} \geq \frac{1}{Nk} \cdot N \cdot \varepsilon^2_A = \varepsilon^2_A/k,\end{aligned}$$

where the first inequality is by Lemma 3.2 (for deterministic algorithms) and last one follows since $\varepsilon_A = \frac{1}{N}\sum_r \varepsilon_{A_r}$ and using the inequality $\|x\|_1 \leq \sqrt{N} \cdot \|x\|_2$, for any $x \in \mathbb{R}^N$.

Continuing the proof Lemma 3.4, we now find a lower bound on the probability of getting a collision, namely a lower bound on the quantity $\mathrm{CP}(f^k(U_n, H^m), H^m)$. We do so by bounding the (possibly smaller) probability of getting a collision $(x_0, \overline{h_0}) = (x_1, \overline{h_1}) \in \mathcal{L}$, for two independent values of $f^k(U_n, H^m), H^m$. We first request that both $(x_0, \overline{h_0}) \in \mathcal{L}$ and $(x_1, \overline{h_1}) \in \mathcal{L}$. This happens with probability at least δ^2. Once inside $\mathcal{L}$, we know that the probability of collision is at least $1/|\mathcal{L}|$. Altogether:

$$\mathrm{CP}(f^k(U_n, H^m), H^m) \geq \delta^2 \cdot \frac{1}{|\mathcal{L}|}. \tag{4}$$

Combining Claim 3.5 and Eq(4), we get $\frac{|\mathcal{L}|}{|\mathcal{H}|^m \cdot |\mathrm{Im}(f)|} \geq \frac{\delta^2}{k}$. Since the probability of getting a value in $\mathcal{L}$ when choosing a random element in $\mathrm{Im}(f) \times \mathcal{H}^m$ is exactly $\frac{|\mathcal{L}|}{|\mathcal{H}|^m \cdot |\mathrm{Im}(f)|}$ (this follows since for a regular function f the distribution $f(U_n)$ is the uniform distribution over $\mathrm{Im}(f)$). It follows that $\Pr[(y, \overline{h}) \in \mathcal{L}] \geq \delta^2/k$ as requested. □

3.3 Pseudorandom Generators from Regular One-Way Functions

After showing that the randomized iterations of a regular one-way function are hard to invert, it is natural to follow the footsteps of the BMY construction to construct a pseudorandom generator. Rather than using simple iterations of the function f, randomized iterations of f are used instead, with fresh randomness in each application. As in the BMY case, a hardcore-bit(s) of the current input is taken at each stage. In order to keep things more readable, we start by giving our pseudorandom generator based on regular one-way functions with super-polynomial hardness (i.e., standard one-way functions). In Section 3.3.1, we generalize this result to regular one-way functions with arbitrary hardness. In particular, we get more efficient pseudorandom generators, assuming that underlying regular one-way functions are exponentially hard.

Theorem 3.6. *Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a regular function, let $\mathcal{H}$ be an efficient family of pairwise-independent length-preserving hash functions over $\{0,1\}^n$ and let $b(\cdot) \stackrel{\text{def}}{=} gl(\cdot)_1$, where gl is the Goldreich-Levin hardcore function (see Theorem 2.12). We define G over $\{0,1\}^n \times \mathcal{H}^n \times \{0,1\}^{2n}$ as*

$$G(x, \overline{h}, r) = (b(f^1(x, \overline{h}), r), \ldots, b(f^{n+1}(x, \overline{h}), r), \overline{h}, r).$$

Assume that f is one way, then G is a pseudorandom generator.

We note that by applying Proposition 2.17 with respect to the above generator, we can get a generator defined on every input length, with similar security and output stretch.

Proof. Denote by $b^k(x, \overline{h}, r) = gl(f^k(x, \overline{h}), r)_1$. Corollary 3.3 yields that $b^k(x, \overline{h}, r)$ is a hardcore function of $\widehat{f^k}(x, \overline{h}, r) = (f^k(x, \overline{h}), \overline{h}, r)$. In the following we show that any algorithm that breaks the pseudorandomness of G, violates the hardness of $b(f^{k-1})$ for some $k \in [n+1]$.

For the proof we consider a reordering of the output bits of G and view it as $(r, \overline{h}, b^{n+1}, \ldots, b^1)$, as this shuffling has no effect the pseudorandomness property. Yao [Yao82] showed using a hybrid argument that it is, up to linear factor, as hard to distinguish a pseudorandom sequence from a random one, as it is to predict the next bit of the sequence for every prefix of the sequence. Thus it suffices to show that for every $k \in [n+1]$ it is hard to predict b^{k-1} given $(r, \overline{h}, b^{n+1}, \ldots, b^k)$. Assume toward a contradiction the existence of a polynomial-time algorithm P for which $\Pr[P(r, \overline{h}, b^{n+1}, \ldots, b^k) = b^{k-1}] > \frac{1}{2} + \phi(n)$ for some non-negligible function $\phi(n)$. Consider the following efficient algorithm M^P for predicting b^{k-1} given $(r, \overline{h}, f^k(x, \overline{h}))$.

Algorithm 3.7. *(M^P)*

Input: $(r, h_1, \ldots, h_{k-1}, f^k(x, h_1, \ldots, h_{k-1}))$.

Operation:

1. *Choose uniformly at random* $h_k, \ldots, h_n \in \mathcal{H}$,
2. *Generate* $f^{k+1}, \ldots, f^{n+1}$ *from* $(f^k, h_{k+1}, \ldots, h_n)$, *i.e.,* $f^{k+j}(x, h_1, \ldots, h_n) = f^j(h_k(f^k(x, h_1, \ldots, h_{k-1})), h_{k+1}, \ldots, h_n)$.
3. *Output* $P(r, h_1 \ldots, h_n, b^{n+1}, \ldots, b^k)$

By choosing $h_{k+1}, \ldots, h_n$ independently at random, M^P generates a series $(f^1, \ldots, f^{n+1})$ that has the same distribution as in the evaluation of G. Thus, the procedure M^P succeeds in predicting b^{k-1} with probability at least $\frac{1}{2} + \phi(n)$, in contradiction to Corollary 3.3. □

3.3.1 From any hardness

The next theorem generalizes Theorem 3.6 for regular one-way function with arbitrary hardness. The proof (omitted) follows the same lines as the proof of Theorem 3.6, using the "any hardness" variant of Corollary 3.3 (see Corollary 4.2 for a generalized version of this variant).

Theorem 3.8. *Let f, $\mathcal{H}$, f^k and gl be as in Theorem 3.6, let $\ell(n) \in [n]$ be a polynomial-time computable function and let $m(n) = \lceil n/\ell \rceil + 1$. We define G over $\{0,1\}^n \times \mathcal{H}^m \times \{0,1\}^{2n}$ as*

$$G(x, \overline{h}, r) = (gl(f^1(x, \overline{h}), r)_{1,\ldots,\ell}, \ldots, gl(f^m(x, \overline{h}), r), \overline{h}, r)_{1,\ldots,\ell}).$$

Assume that f is $(T(n), \varepsilon(n))$-one-way, then G is a $(T(n) \cdot 2^{-\ell(n)} \cdot (\varepsilon(n)/n)^{O(1)}, \varepsilon(n)/m)$-pseudorandom generator.

For exponentially hard one-way regular one-way function, combining the above theorem and Proposition 2.17 yields the following "linear stretch" generator:

Corollary 3.9. *Assume there exists a regular exponentially hard one-way function, then there exists a $2^{\Omega(n)}$-pseudorandom generator from $\{0,1\}^n$ to $\{0,1\}^{n+\Omega(n)}$.*

3.4 An Almost-Linear-Input Generator from Regular One-Way Functions

Assuming that the underlying function is one way in the usual sense (i.e., of super-polynomial hardness), the pseudorandom generator presented in the previous section (when using Theorem 3.8) stretches a seed of length $\Theta(n^2/\log(n))$ by $\log(n)$ bits. Although this is an improvement over the GKL generator, it still translates to a rather high loss of security, since the security of the generator on $d(n)$ bits relies on the security of regular one-way function on $\sqrt{d(n)}$ bits. In this section we give a modified construction of the pseudorandom generator of Theorem 3.6 that takes a seed of length only $d(n) \in \Theta(n \log n)$.

Notice that the input length of the generator of Theorem 3.6 is dominated by the description of the n independent hash functions that define $\overline{h} = (h_1, \ldots, h_n)$. The idea of the new construction is to derandomize the choice of $\overline{h}$. Thus, $h_1, \ldots, h_n$ are no longer chosen independently, but are chosen in a way that is sufficient for the proof to go through. The derandomization uses generators

against bounded-space distinguishers, and specifically we can use the generator of Nisan [Nis92], (or that of Impagliazzo, Nisan and Wigderson [INW94]). The key observation is that calculating the randomized iterate of an input can be viewed as a bounded-space algorithm, alternatively presented here as a bounded-width layered branching program. More accurately, at each step the branching program gets a random input h_i and produces $f^{i+1} = f(h_i(f^i))$. We will show that indeed when replacing $h_1, \ldots, h_n$ with the output of a generator that fools such branching programs, then the proof of security still holds (and specifically the proof of Lemma 3.4).

Theorem 3.10. *Let f, $\mathcal{H}$, f^k and b be as in Theorem 3.6. Let $v(\mathcal{H}) = 2n$ be the description length of $h \in \mathcal{H}$ and let $\mathrm{BSG} : \{0,1\}^{q(n) \in \Theta(n \log n)} \mapsto \{0,1\}^{n \cdot v(\mathcal{H})}$ be a bounded-space generator that 2^{-n}-fools every $(2n, n, v(\mathcal{H}))$-LBP.[19] We define G over $\{0,1\}^n \times \{0,1\}^{q(n)} \times \{0,1\}^{2n}$ as*

$$G(x, s, r) = (b(f^1(x, \mathrm{BSG}(s)), r), \ldots, b(f^{n+1}(x, \mathrm{BSG}(s)), r), s, r).$$

Assuming that f is one-way, then G is a pseudorandom generator.

As in the case of Theorem 3.6, by applying Proposition 2.17 with respect to the above generator, we can get a generator defined on every input length, with similar security and output stretch.

Proof outline. The proof of the derandomized version follows in the steps of the proof of Theorem 3.6. We give a high-level outline of this proof, focusing only on the main technical lemma that changes slightly. The proof first shows that given the k'th randomized iterate $f^k(x, \overline{h})$ and $\overline{h}$ it is hard to compute $f^{k-1}(x, \overline{h})$ (analogously to Lemma 3.2), only now this also holds when the hash functions are chosen as the output of the bounded-space generator. The proof is identical to the proof of Lemma 3.2, only replacing appearances of $\overline{h}$ with the seed s. Again, the key to the proof is the following technical lemma (slightly modified from Lemma 3.4):

Lemma 3.11. *For every set $\mathcal{L} \subseteq \mathrm{Im}(f) \times \{0,1\}^{q(n)}$, if*

$$\Pr[(f^k(U_n, \mathrm{BSG}(U_{q(n)})), U_{q(n)}) \in \mathcal{L}] \geq \delta,$$

then

$$\Pr[(f(U_n), U_{q(n)}) \in \mathcal{L}] \geq \delta^2/(k+1).$$

Once we know that $f^{k-1}(x, \mathrm{BSG}(s))$ is hard to compute given $f^k(x, \mathrm{BSG}(s))$ and s (for random x and s), we deduce that one cannot predict a hardcore bit $b(f^{k-1}(x, \mathrm{BSG}(s)), r)$ given $f^k(x, \mathrm{BSG}(s))$ and the seed s to the bounded-space generator. From here, the proof follow just as the proof of Theorem 3.6 in showing that the output of G is an unpredictable sequence and therefore a pseudorandom sequence.

Proof. (of Lemma 3.11) Denote by $g : \{0,1\}^n \times \{0,1\}^{q(n)} \mapsto \{0,1\}^n \times \{0,1\}^{q(n)}$ the function taking inputs of the form (x, s) to outputs of the form $(f^k(x, \mathrm{BSG}(s)), s)$. We proceed by giving bounds on the collision probability of g. For every two inputs (x_0, s_0) and (x_1, s_1) to g, in order to have a collision we must first have that $s_0 = s_1$ which happens with probability $1/2^{q(n)}$. Now, given that $s_0 = s_1 = s$ (with a random s), we analyze the probability of the event that $f^k(x_0, \mathrm{BSG}(s))$ equals $f^k(x_1, \mathrm{BSG}(s))$.

[19] Such generators follow from Theorem 2.6.

Consider the following $(2n, n, v(\mathcal{H}))$-LBP M for the input pair (x_0, x_1): the source node in labeled by $(y_1^0 = f(x_0), y_1^1 = f(x_1))$, and being on node labeled by (y_i^0, y_i^1) in the i'th layer, it does the following on input $h \in \mathcal{H}$: let $y_{i+1}^0 = f(h(y_i^0))$ and $y_{i+1}^1 = f(h(y_i^0))$. If $i < m$, it moves to the node (y_{i+1}^0, y_{i+1}^1) in the $i+1$ layer. Otherwise (i.e., $i = m-1$), it moves to the 1-labeled node (in the $m+1$'th layer) in case $y_{m+1}^0 = y_{m+1}^1$ and to the 0-labeled node otherwise.

The LBP described above has parameters $s = 2n$, $m = n$ and $v = 2n$. Furthermore, it accepts (outputs 1) with probability that is exactly the desired collision probability, that is, the probability that $f^k((x_0, \overline{h})) = f^k((x_1, \overline{h}))$ over *any* distribution on $\overline{h} = (h_1, \ldots, h_{k-1})$. For every pair (x_0, x_1) with $f(x_0) \neq f(x_1)$, this probability over random $\overline{h}$ was bounded in the proof of Lemma 3.4 by:

$$\Pr_{\overline{h} \leftarrow \mathcal{H}^{k-1}}[f^k(x_0, \overline{h}) = f^k(x_1, \overline{h})] \leq \frac{k-1}{|\mathrm{Im}(f)|}$$

Since the generator fools the above LBP, then replacing the random inputs $\overline{h}$ with the output of the bounded-space generator does not change the probability of acceptance by more than $\varepsilon_{\mathrm{BSG}}(n) = 2^{-n}$. Therefore, assuming $f(x_0) \neq f(x_1)$, we have that

$$\Pr_{s \leftarrow U_{q(n)}}[f^k(x_0, \mathrm{BSG}(s)) = f^k(x_1, \mathrm{BSG}(s))] \leq \frac{k-1}{|\mathrm{Im}(f)|} + \frac{1}{2^n}$$

When taking the probability over random (x_0, x_1), we also add the probability that $f(x_0) = f(x_1)$. Thus,

$$\Pr_{x_0, x_1 \leftarrow U_n, s \leftarrow U_{q(n)}}[f^k(x_0, \mathrm{BSG}(s)) = f^k(x_1, \mathrm{BSG}(s))] \leq \frac{k}{|\mathrm{Im}(f)|} + \frac{1}{2^n} \leq \frac{k+1}{|\mathrm{Im}(f)|}$$

When the above is plugged into the calculation of the collision probability of the function g (recall that $g(x, s) = (f^k(x, \mathrm{BSG}(s)), s)$), we get:

$$\mathrm{CP}(g(U_n, U_{q(n)})) \leq \frac{k+1}{2^{q(n)} \cdot |\mathrm{Im}(f)|} \tag{5}$$

Continuing the proof, we now find a lower bound on the probability of getting a collision inside the set $\mathcal{L}$, which is a lower bound on the probability of getting a collision at all. We first request that both $(x_0, s_0) \in \mathcal{L}$ and $(x_1, s_1) \in \mathcal{L}$, which happens with probability at least δ^2. Once inside $\mathcal{L}$, we know that the probability of collision is at least $1/|\mathcal{L}|$. Altogether:

$$\mathrm{CP}(g(U_n, U_{q(n)})) \geq \delta^2 / |\mathcal{L}| \tag{6}$$

Combining Eq(5) and Eq(6), we get

$$\frac{|\mathcal{L}|}{2^{q(n)} |\mathrm{Im}(f)|} \geq \frac{\delta^2}{k+1}.$$

But the probability of getting a value in $\mathcal{L}$ when choosing a random element in $\mathrm{Im}(f) \times \{0,1\}^{q(n)}$ is exactly $\frac{|\mathcal{L}|}{2^{q(n)} \cdot |\mathrm{Im}(f)|}$. Thus, $\Pr[(y, s) \in \mathcal{L}] \geq \delta^2/(k+1)$ as requested. □

Remark 3.12. *It is tempting to think that one should replace Nisan/Impagliazzo-Nisan-Wigderson generator in the above proof with the generator of Nisan and Zuckerman [NZ96]. That generator may have seed of size $\Theta(n)$ (rather than $\Theta(n \log n)$), where $s = 2n$ as in our case. Unfortunately,*

with such a short seed, that generator incurs an error $\varepsilon_{\mathrm{BSG}}(n) = 2^{-n^{1-\gamma}}$ *for some constant* γ*, which is too high for our proof to work. In order for the proof to go through we need that* $\varepsilon_{\mathrm{BSG}}(n) < poly(n)/|\mathrm{Im}(f)|$*. Interestingly, this means that we get a linear-input construction when the image size is significantly smaller than* 2^n*; in order to achieve a linear-input construction in the general case, we need better generators against LBPs (that have both short seed and small error).*

3.4.1 Dealing with non length-preserving functions

The pseudorandom generators presented in this section assumed that the underlying regular one-way function is length preserving. We mention that this is not a necessity and outline how any regular one-way function can be used. For the simple case that f is shrinking, simply padding the output to the same length is sufficient. The more interesting case is of a length-expanding one-way function f. The important point is that we want the generator to be almost linear in the length of the input to f rather than its output. In Lemma 2.9 we show how to transform an expanding one-way function f from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$ into a length preserving one-way function from $\{0,1\}^{d(n)}$ to $\{0,1\}^{d(n)}$ for some $d(n) \in \Theta(n)$. This construction, however, does not maintain the regularity of the one-way function (it maintains only an approximate regularity).

For the regular case we suggest a different solution; rather than changing the underlying one-way function to be length preserving, we change the randomizing hash functions to be shrinking. That is, given a regular one-way function $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, define the randomized iterate of this function with respect to a family of hash functions from $\{0,1\}^{\ell(n)}$ to $\{0,1\}^n$. The randomized iterate is now well defined, and all the proofs given above for length preserving one-way functions, readily hold for this new construction. The only problem with this approach is that the description of such hash functions is too long (it is $\Theta(\ell(n))$ instead of $\Theta(n)$). This is overcome by using an efficient family of *almost* pairwise-independent hash functions from $\ell(n)$ bits to n bits with error 2^{-n} (see Definition 2.2), which requires a description of only $\Theta(n)$ bits.

4 Pseudorandom Generators from Exponentially Hard One-Way Functions

4.1 Overview

4.1.1 The randomized iterate and general one-way functions

As mentioned in the introduction, the last randomized iteration of a general one-way function is not necessarily hard to invert, and in fact may be easy to invert. This hardness, however, is not totally diminished, it simply deteriorates in every additional iteration. By refining the techniques used in the case of regular one-way functions, we manage to give a lower bound on this deterioration. More precisely, we show in Section 4.2 that there exists a set $\mathcal{S}$ of inputs to f^k with density at least $\frac{1}{k}$, such that the k'th randomized iteration is hard to invert over inputs taken from $\mathcal{S}$. As a result, a hard core bit of the k'th randomized iteration has the guarantee that with probability at least $\frac{1}{k}$ it is pseudorandom.

4.1.2 The multiple randomized iterate

Our goal is to get a string of pseudorandom bits, and the idea is to run t independent copies of the randomized iterate (on t independent inputs). We call this the *multiple randomized iterate*. From each of the t copies, we output a hardcore bit of the k'th iteration. This forms a string of t bits, of which $\frac{t}{k}$ are expected to be random looking. The next step is to run a randomness extractor on such a string (where the output of the extractor is of length, say, $\frac{t}{2k}$). This ensures that with very high probability, the output of the extractor is a pseudorandom string of bits.

4.1.3 The pseudorandom generator – a first attempt

A first attempt for the pseudorandom generator runs the multiple randomized iterate (on t independent inputs) for m (to be determined later) iterations. For each $k \in [m]$, we extract $\frac{t}{2k}$ bits at the k'th iteration. These bits are guaranteed to be pseudorandom (even when given all of the values at the $(k+1)$'th iterate and all of the randomizing hash functions). Thus, outputting the concatenation of the pseudorandom strings for the different values of k forms a long pseudorandom output (by a standard hybrid argument).

This concatenation, however, is still not long enough. It is required that the output of the generator is longer than its input, which is not the case here. The input contains t strings $x_1, \ldots, x_t$ and tm hash functions. The hash functions are included in the output, so the rest of the output needs to make up for the tn bits of $x_1, \ldots, x_t$. At each iteration we output $\frac{t}{2k}$ bits which adds up to $\sum_{k=1}^{m} \frac{t}{2k}$ bits. This harmonic progression is bounded by $t \cdot \frac{\log m}{2}$, Thus, in order to exceed the tn loss bits of the input, we need $m > 2^n$ that is far from being efficient.

4.1.4 The pseudorandom generator and exponential hardness

The failed generator from above can be remedied when the exponential hardness comes into play. It is known that if a function is 2^{cn}-one-way (for some constant $c \in (0,1)$), then it has a $2^{c'n}$-hardcore function of $c'n$ bits (for another constant c'). Thus, if the original hardness was exponential, then in the k'th iteration we can actually extract $c'n$ random looking strings, each of length $\frac{t}{2k}$. Altogether, we get that the output length is $c'n \cdot \sum_{k=1}^{m} \frac{t}{2k} \geq c'tn \cdot \log m$. Thus, for a choice of m such that $\log m > c'$, we get that the overall output is a pseudorandom string of length greater than the input. It follows that to get a $T(n)$-pseudorandom generator, the input length of the above generator is $\Theta(tn)$, where t can be taken in $\Theta(\log(m \cdot T(n)))$. In particular, in order to get a $2^{\Omega(n)}$-pseudorandom generator, one needs to take a seed of length $\Theta(n^2)$.

To sum up, we describe the full construction in a slightly different manner: one first creates a matrix of size $t \times m$, where each *row* in the matrix is generated by computing the first m randomized iterates of f (each row takes independent inputs). Now from each entry in the matrix $\Theta(cn)$ hardcore bits are computed (thus generating a matrix of hardcore bits). The final stage runs a randomness extractor on each of the *columns* of the hardcore bits matrix.[20] Moreover, the number of pseudorandom bits extracted from a column deteriorates from one iteration to another ($\frac{t}{k}$ pseudorandom bits are taken at the columns associated with the k'th randomized iterate).

[20] Note that each execution of the extractor runs on a column in which each entry consists of a single bit (rather than $\Theta(cn)$ bits). In other words, we translate each column of $\Theta(cn)$ bit-strings to $\Theta(cn)$ bit columns. This is a requirement of the proof technique.

4.1.5 Some remarks

- Our method also works for $2^{\phi(n)}$-one-way functions only as long as $\phi(n) \in \Omega(\frac{n}{\log n})$. Our method fails to handle smaller values of ϕ, since as m grows, the value $\frac{1}{m}$ becomes too small, requiring t to grow substantially.
- The description in this section focuses on length-preserving one-way functions, the results can be generalized, using Lemma 2.9, to use non-length preserving functions.

4.2 The Last Randomized Iterate is (sometimes) Hard to Invert

We now formally state and prove the key observation of this section; there exists a set of inputs of significant weight for which it is hard to invert the k'th randomized iteration, even if given access to all of the hash functions leading up to this point. This statement is a generalization of Lemma 3.2 to all functions rather than just regular ones. In the following Lemma it is instructive to ignore the parameter gap (set $\text{gap}(n) = 0$), as it will only be used in Section 5.

Lemma 4.1. *Let f, m, k, $\mathcal{H}$ and f^k be as in Definition 3.1, let $\text{gap}(n) \in \{0,\ldots,n\}$ and define $\{\mathcal{S}_n \subseteq \{0,1\}^n \times \mathcal{H}^m\}_{n\in\mathbb{N}}$ as*

$$\mathcal{S}_n = \left\{(x,\overline{h}) \in \{0,1\}^n \times \mathcal{H}^m \colon \mathrm{D}_f(f^k(x,\overline{h})) + \text{gap}(n) \geq \max_{j\in[k]} \mathrm{D}_f(f^j(x,\overline{h}))\right\}.$$

The following hold for every $n \in \mathbb{N}$:

1. $\Pr[(U_n, H^m) \in \mathcal{S}_n] \geq 1/k$.
2. *For any algorithm A with*

$$\varepsilon_A = \Pr[A(f^k(U_n,H^m),H^m) = f^{k-1}(U_n,H^m) \wedge (U_n,H^m) \in \mathcal{S}_n],$$

it holds that

$$\Pr[H^m_{k-1}(A(f(U_n),H^m)) \in f^{-1}(f(U_n)) \wedge (U_n,H^m) \in \mathcal{S}_n] \geq \varepsilon_A^2/nk \cdot 2^{\text{gap}(n)+1}.$$

Note that by considering a function $\text{gap} > 0$, one can increase the size of $\mathcal{S}_n$ at the price of weakening the hardness of f^k over it. We will use this feature in Section 5, in which we will set $\text{gap} = \Theta(\log n)$, whereas in the current section we only make use of the choice $\text{gap} = 0$.

As in Section 3, assuming that f is a one-way function yields the following corollary:

Corollary 4.2. *Let let f, m, k, $\mathcal{H}$, f^k, gap and $\mathcal{S}_n$ be as in Lemma 4.1. Assume that f is a $(T(n),\varepsilon(n))$-one-way function and that $m(n) \leq \text{poly}(n)$, then the following hold for $\varepsilon'(n) = n^{O(1)} \cdot \sqrt{2^{\text{gap}(n)} \cdot \varepsilon(n)}$:*

1. $\Pr[A(f^k(U_n,H^m),H^m) = f^{k-1}(U_n,H^m) \mid (U_n,H^m) \in \mathcal{S}_n] \leq \varepsilon'(n)$ *for any algorithm A of running time $T(n) - n^{O(1)}$ and large enough n, and*
2. *For any $\ell(n) \in [n]$, the function $hc^k \colon \{0,1\}^n \times \mathcal{H}^m \times \{0,1\}^{2n} \mapsto \{0,1\}^\ell$ defined as $hc^k(x,\overline{h},r) = gl(f^{k-1}(x,\overline{h}),r)_\ell$ is a $(T(n) \cdot (\varepsilon(n)/n)^{O(1)}, 2^\ell \cdot \varepsilon'(n))$-hardcore function of $\widehat{f^k}(x,\overline{h},r) = (f^k(x,\overline{h}),\overline{h},r)$ over $\mathcal{S}_n$, where gl is the Goldreich-Levin hardcore function (see Theorem 2.12).*

Proof. (of Corollary 4.2) Lemma 4.1(1) yields that $\Pr[(U_n, H^{k-1}) \in \mathcal{S}_n] \geq 1/k$. Hence, for any algorithm A such that $\varepsilon'(n) = \Pr[A(f^k(U_n, H^m), H^m) = f^{k-1}(U_n, H^m) \mid (U_n, H^m) \in \mathcal{S}_n] > n^{O(1)} \cdot \sqrt{2^{\mathrm{gap}(n)} \cdot \varepsilon(n)}$, it holds that

$$\Pr[A(f^k(U_n, H^m), H^m) = f^{k-1}(U_n, H^m) \wedge (U_n, H^m) \in \mathcal{S}_n] > \varepsilon'(n)/k > n^{O(1)} \cdot \sqrt{2^{\mathrm{gap}(n)} \cdot \varepsilon(n)},$$

and the proof of the corollary follows from Lemma 4.1(2), using analogous lines to that used to derive Corollary 3.3 from Lemma 3.2. □

Proof. (of Lemma 4.1) The pairwise independence of $\mathcal{H}$ yields that the value $f^i(U_n, H^m)$ is independently and randomly chosen from the distribution $f(U_n)$ for every $i \in [m+1]$. Thus, a symmetry argument yields that the k'th iteration has the heaviest preimage size with probability at least $1/k$, which yields the first part of the lemma. Where the second part of the lemma immediately follows from the following lemma (see the proof of Lemma 3.2):

Lemma 4.3. *Let f, m, k, f^k, gap and $\mathcal{S}_n$ be as in Lemma 4.1. Then for every $n \in \mathbb{N}$ and $\mathcal{L} \subseteq \mathcal{S} = \mathcal{S}_n$, if*

$$\Pr[(f^k(U_n, H^m), H^m) \in \mathcal{L}] \geq \delta,$$

then

$$\Pr[(f(U_n), H^m) \in \mathcal{L}] \geq \frac{\delta^2}{nk \cdot 2^{\mathrm{gap}+1}}.$$

□

Proof. (of Lemma 4.3) Divide the outputs of the function f into n slices according to their preimage size. The set $\mathcal{L}$ is divided accordingly into n subsets. For every $j \in [n]$, define the j'th slice $\mathcal{L}_j = \left\{(z, \overline{h}) \in \mathcal{L} \mid \mathrm{D}_f(z) = j\right\}$. We divide $\mathcal{S}$ into corresponding slices as well; define the j'th slice as $\mathcal{S}_j = \left\{(x, \overline{h}) \in \mathcal{S} \mid \mathrm{D}_f(f^k(x, \overline{h})) = j\right\}$ (note that since $\mathcal{S}_j \subseteq \mathcal{S}$, it holds for each $(x, \overline{h}) \in \mathcal{S}_j$ and $0 \leq i < k$, that $\mathrm{D}_f(f^i(x, \overline{h})) \leq \mathrm{D}_f(f^k(x, \overline{h})) + \mathrm{gap} = j + \mathrm{gap}$). The proof of Lemma 4.3 follows the methodology of the analogous lemma for the regular case (Lemma 3.4). More precisely, the proof studies the collision probability of f^k, only here we look at f^k when restricted to $\mathcal{S}_j$ (i.e., we work separately on each slice). Denote this as:

$$\mathrm{CP}(f^k(U_n, H^m) \wedge \mathcal{S}_j) \stackrel{\mathrm{def}}{=} \Pr[(f^k(x_0, \overline{h}_0), \overline{h}_0) = (f^k(x_1, \overline{h}_1), \overline{h}_1) \wedge (x_0, \overline{h}_0), (x_1, \overline{h}_1) \in \mathcal{S}_j],$$

where $(x_0, \overline{h}_0)$ and $(x_1, \overline{h}_1)$ are iid over $\{0,1\}^n \times \mathcal{H}^m$. We first give an upper-bound on this collision probability.

Claim 4.4.

$$\mathrm{CP}(f^k(U_n, H^m) \wedge \mathcal{S}_j) \leq \frac{k}{|\mathcal{H}|^m \cdot 2^{n-j}}$$

Proof. For every two inputs $(x_0, \overline{h}_0)$ and $(x_1, \overline{h}_1)$ in $\{0,1\}^n \times \mathcal{H}^m$, in order to have a collision we must first have that $\overline{h}_0 = \overline{h}_1$, which happens with probability $(1/|\mathcal{H}|)^k$. Given that $\overline{h}_0 = \overline{h}_1 = \overline{h}$ (with $\overline{h} \in \mathcal{H}^m$ being uniform), we require also that $f^k(x_0, \overline{h})$ equals $f^k(x_1, \overline{h})$.

If $f(x_0) = f(x_1)$, then a collision is assured. Since it is required that $(x_0, \overline{h}_0) \in \mathcal{S}_j$, it holds that $\mathrm{D}_f(f(x_0)) \leq \mathrm{D}_f(f^k(x_0, \overline{h})) + \mathrm{gap} = j + \mathrm{gap}$ and therefore $\left|f^{-1}(f(x_0))\right| \leq 2^{j+\mathrm{gap}}$. Thus, the probability for that $x_1 \in f^{-1}(f(x_0))$ (and thus of $f(x_0) = f(x_1)$) is at most $2^{j+\mathrm{gap}-n}$. Otherwise,

there must be an $j \in [k-1]$ for which $f^j(x_0, \overline{h}) \neq f^j(x_1, \overline{h})$, but $f^{j+1}(x_0, \overline{h}) = f^{j+1}(x_1, \overline{h})$. Since $f^j(x_0, \overline{h}) \neq f^j(x_1, \overline{h})$, the pairwise independence of $\mathcal{H}$ yields that the values of $\overline{h}_j(f^j(x_0, \overline{h}))$ and $\overline{h}_j(f^j(x_1, \overline{h}))$ are uniformly random values in $\{0,1\}^n$. Thus, $f(\overline{h}_j(f^j(x_0, \overline{h}))) = f(\overline{h}_j(f^j(x_1, \overline{h})))$ also happens with probability at most $2^{j+\text{gap}-n}$. Altogether,

$$\text{CP}(f^k(U_n, H^m) \wedge \mathcal{S}_j) \leq k \cdot \frac{2^{j+\text{gap}-n}}{|\mathcal{H}|^m} = \frac{k}{|\mathcal{H}|^m \cdot 2^{n-\text{gap}-j}}.$$

□

Continuing the proof Lemma 4.3, we now give a lower-bound for the above collision probability. We seek the probability of getting a collision inside $\mathcal{S}_j$ and further restrict our calculation to collisions whose output lies in the set $\mathcal{L}_j$ (this further restriction may only reduce the collision probability and thus the lower bound holds also without the restriction). For each slice, denote $\delta_j = \Pr[(f^k(x, \overline{h}), \overline{h}) \in \mathcal{L}_j \wedge (x, \overline{h}) \in \mathcal{S}_j]$. In order to have this kind of collision, we first request that both inputs are in $\mathcal{S}_j$ and generate outputs in $\mathcal{L}_j$, which happens with probability δ_j^2. Once inside $\mathcal{L}_j$, we require that both outputs collide (which happens with probability at least $\frac{1}{|\mathcal{L}_j|}$). Altogether:

$$\text{CP}(f^k(U_n, H^m) \wedge \mathcal{S}_j) \geq \delta_j^2 / |\mathcal{L}_j| \tag{7}$$

Combining Claim 4.4 and Eq(7) we get:

$$\frac{|\mathcal{L}_j| \cdot 2^{j+\text{gap}-n}}{|\mathcal{H}|^m} \geq \delta_j^2 / k \tag{8}$$

Note that when taking a random output $z \in Im(f)$ and independent hash functions $\overline{h}$, the probability of hitting an element in $\mathcal{L}_j$ is at least $2^{j-n-1} / |\mathcal{H}|^m$ (since each output in $\mathcal{L}_j$ has preimage at least 2^{j-1}). This means that

$$\Pr[(f(U_n), H^m) \in \mathcal{L}_j] \geq |\mathcal{L}_j| \cdot 2^{j-n-1} / |\mathcal{H}|^m , \tag{9}$$

and by Eq(8) we deduce that $\Pr[(f(U_n), H^m) \in \mathcal{L}_j] \geq \frac{\delta_j^2}{k \cdot 2^{\text{gap}+1}}$. Finally, the probability of hitting $\mathcal{L}$ is $\Pr[(f(U_n), H^m) \in \mathcal{L}] = \sum_j \Pr[(f(U_n), H^m) \in \mathcal{L}_j] \geq \sum_j \frac{\delta_j^2}{k \cdot 2^{\text{gap}+1}}$. Since $\sum_j \delta_j^2 \geq (\sum_j \delta_j)^2 / n$ and (by definition) $\sum_j \delta_j = \delta$, it holds that $\Pr[(f(U_n), H^m) \in \mathcal{L}] \geq \frac{\delta^2}{nk \cdot 2^{\text{gap}+1}}$, as claimed. □

4.3 The Multiple Randomized Iterate

In this section we consider the function $f^{t \times k}$ that consists of t independent copies of the randomized iterate f^k.

Definition 4.5 (the k'th multiple randomized iterate)**.** *Let f, m, k, f^k and $\mathcal{H}$ be as in Definition 3.1. For $t(n) \in \mathbb{N}$, we define the k'th* Multiple Randomized Iterate *$f^{t \times k} : \{0,1\}^{t \times n} \times \mathcal{H}^{t \times m} \mapsto \{0,1\}^{t \times n}$ as:*

$$f^{t \times k}(\overline{x}, V) = (f^k(\overline{x}_1, V_1), \ldots, f^k(\overline{x}_t, V_t)),$$

where $\overline{x} \in \{0,1\}^{t \times n}$ and $V \in \mathcal{H}^{t \times m}$. In the following we denote by $H^{t \times m}$ the random variable uniformly distributed over $\mathcal{H}^{t \times m}$.

For each of the t outputs of $f^{t\times k}$, we look at its hardcore function hc^k. Corollary 4.2 yields that t/k of these t hardcore strings are expected to fall inside the "hard-set" of f^k (and thus are indeed pseudorandom given $(f^{t\times k}(\overline{x},V),V)$). The next step is to invoke a randomness extractor on a concatenation of one bit from each of the different independent hardcore strings. The output of the extractor is taken to be of length $\lfloor \frac{t}{4k} \rfloor$. The intuition being that with high probability, the concatenation of these bits from the different outputs of hc^k contains "pseudoentropy" of at least $\frac{t}{2k}$ bits. Thus, the output of the extractor forms a pseudorandom string and might serve as a hardcore function of the multiple randomized iterate $f^{t\times k}$.

Lemma 4.6 (hardcore function for the multiple randomized iterate)**.** *Let $k(n) < m(n)$, $\ell(n) \leq n$ and $t(n)$ be integer functions. Let* $\mathrm{Ext} : \{0,1\}^{2n} \times \{0,1\}^t \mapsto \{0,1\}^t$ *be such that* $\mathrm{Ext}_i(\cdot) \stackrel{\text{def}}{=} \mathrm{Ext}(\cdot)_{1,\ldots,\lfloor \frac{t}{j} \rfloor}$ *is a $(\lfloor \frac{t}{2j} \rfloor, 2^{-\Omega(\min\{n,t/j\})})$-strong extractor for every $j \in \mathbb{N}$,*[21] *and for $f : \{0,1\}^n \mapsto \{0,1\}^n$ let $f^{t\times k}$ be as in Definition 4.5 and let $hc^k : \{0,1\}^n \times \mathcal{H}^m \times \{0,1\}^{2n} \mapsto \{0,1\}^\ell$ as in Corollary 4.2. We define $hc^{t\times k}\colon \{0,1\}^{t\times n} \times \mathcal{H}^{t\times m} \times \{0,1\}^{4n} \mapsto \{0,1\}^{\ell\cdot\lfloor \frac{t}{4k} \rfloor}$ as*

$$hc^{t\times k}(\overline{x},V,r,s) = (w_1(\overline{x},V,r,s),\ldots,w_\ell(\overline{x},V,r,s)),$$

where $\overline{x} \in \{0,1\}^{t\times n}$, $V \in \mathcal{H}^{t\times m}$, $r,s \in \{0,1\}^{2n}$ and $w_i(\overline{x},V,r,s) \stackrel{\text{def}}{=} \mathrm{Ext}_k(s,(hc^k(\overline{x}_1,V_1),r)_i,\ldots,(hc^k(\overline{x}_t,V_t),r)_i))$ for every $i \in [\ell]$, and let

$$\widehat{f^{t\times k}}(\overline{x},V,r,s) = (f^{t\times k}(\overline{x},V),V,r,s).$$

Assume that $m(n),t(n) \leq \mathrm{poly}(n)$ and that f is a $(T(n),\varepsilon(n))$ one-way function, where $2^{-\min\{n,t/k^2\}} \leq \varepsilon(n) < 2^{-\ell}\cdot n^{-O(1)}$ is polynomial-time computable, then $hc^{t\times k}$ is a $(T(n)\cdot \varepsilon(n)^{O(1)}, \varepsilon(n)^{\Omega(1)})$-hardcore function of $\widehat{f^{t\times k}}$.

Proof. Corollary 4.2 yields that hc^k is a $(T\cdot\varepsilon^{O(1)}, 2^\ell\cdot n^{O(1)}\cdot\sqrt{\varepsilon})$ hardcore function of $\widehat{f^k}$ over a set $\mathcal{S}_n$, where $\mathcal{S}_n$ is of density $1/k$ over $\{0,1\}^{t\times n}\times\mathcal{H}^{t\times m}\times\{0,1\}^{4n}$. Since $\varepsilon < 2^{-\ell}\cdot n^{-O(1)}$, it follows that hc^k is a $(T\cdot\varepsilon^{O(1)}, (1-\frac{1}{2k})/2)$ hardcore function of $\widehat{f^k}$ (over the whole domain). Let $j = j(n) \in [\ell]$, a straightforward hybrid argument yields that the predicate $hc^k_j(\cdot) \stackrel{\text{def}}{=} hc^k(\cdot)_j$ is a $(T''(n) = T'(n)/n^{O(1)}, (1-\frac{1}{3k})/2)$ hardcore predicate of $g^k_i(\cdot) \stackrel{\text{def}}{=} (\widehat{f^k}(\cdot), hc^k(\cdot)_{1,\ldots,j-1})$. In the following we prove that $hc^{t\times k}_j(\cdot) \stackrel{\text{def}}{=} hc^{t\times k}(\cdot)_j$ is a $(T\cdot(\varepsilon)^{O(1)}, \varepsilon^{\Omega(1)})$ hardcore predicate of $g^{t\times k}_i(\cdot) \stackrel{\text{def}}{=} (\widehat{f^{t\times k}}(\cdot), hc^k(\cdot)_{1,\ldots,j-1})$, and the proof of the lemma would follow via a standard hybrid argument.

Assuming that the above does not hold, Lemma 2.14 yields that there exists an oracle-aided algorithm D of running time $T\cdot\varepsilon^{O(1)}$ such that

$$\Delta^{D^{g^k_{j,n},hc^k_{j,n}}}\Big((g^k_j(X),hc^k_j(X)),(g^k_j(X),U)\Big) > (1-\frac{1}{3k})/2,$$

for infinitely many n's, where X and U are uniformly distributed over $\mathrm{Dom}(g^k_{j,n})$ (i.e., $\{0,1\}^n \times \mathcal{H}^{m(n)} \times \{0,1\}^{2n}$) and $\{0,1\}$ respectively, and $g^k_{j,n}$ and $hc^k_{j,n}$ are the restrictions of g^k_j and hc^k_j to inputs inside $\mathrm{Dom}(g^k_{j,n})$. The point is that given $y \in \mathrm{Im}(g^k_{j,n})$, the function $g^k_{j,n}$ and $hc^k_{j,n}$ are

[21] For example, the leftover hash lemma ([BBR85, IL89]) yields that $\mathrm{Ext}(h,x) = h(x)$, where h if is a member a family of pairwise independent hash functions from $\{0,1\}^t$ to $\{0,1\}^t$, is a good choice.

efficiently computable (even thought they might not be so without this advice). Hence, the above yields that

$$\Delta^{D'}\left((g_j^k(X), hc_j^k(X)), (g_j^k(X), U)\right) > (1 - \frac{1}{3k})/2,$$

for some algorithm D' of running time $T \cdot \varepsilon^{O(1)}$, in contradiction to hardness of hc_j^k with respect to g_j^k. □

4.4 A Pseudorandom Generator from Exponentially Hard One-Way Functions

We are now ready to present our pseudorandom generator. After deriving a hardcore function for the multiple randomized iterate, the generator is similar to the construction from regular one-way function. That is, run randomized iterations and output hardcore bits. The major difference in our construction is that, for starters, it uses hardcore functions rather than hardcore bits. More importantly, the amount of hardcore bits extracted at each iteration is not constant and deteriorates with every additional iteration. As in Section 3, our generator applies the standard BMY principle on the multiple randomized iterate, outputting the hardcore bits of each iteration.

Our generator is given in the following theorem, whose proof immediately follows by Lemma 4.6 and a standard BMY like indistinguishability argument (cf., the proof of Theorem 3.6).

Theorem 4.7. *Let m, ℓ, t, $hc^{t\times k}$ and $f^{t\times k}$ be as in Lemma 4.6. We define $G\colon \{0,1\}^{t\times n} \times \mathcal{H}^{t\times m} \times \{0,1\}^{4n} \mapsto \{0,1\}^{\ell\cdot\sum_{k\in[m+1]}\lfloor\frac{t}{4k}\rfloor} \times \mathcal{H}^{t\times m} \times \{0,1\}^{4n}$ as*

$$G(\overline{x}, V, r, s) = (hc^{t\times 1}(\overline{x}, V, r, s)\ldots, hc^{t\times m}(\overline{x}, V, r, s), V, r, s),$$

where $\overline{x} \in \{0,1\}^{t\times n}$, $V \in \mathcal{H}^{t\times(m-1)}$, $r \in \{0,1\}^{2n}$ and $s \in \{0,1\}^{2n}$.

Assume that $tn < \ell\cdot\sum_{k\in[m+1]}\lfloor\frac{t}{4k}\rfloor$ and that f is (T,ε)-one-way function, where $2^{-\min\{n,t/m^2\}} \leq \varepsilon < 2^{-\ell}\cdot n^{-O(1)}$ is polynomial-time computable, then G is a $(T\cdot\varepsilon^{O(1)}, \varepsilon^{\Omega(1)})$-pseudorandom generator.

Taking $t(n) = n$, $\ell(n) \in \Omega(1)$ and large enough constant m, the above theorem together with Proposition 2.17 yield the following pseudorandom generator from exponentially hard one-way functions:

Corollary 4.8. *Assume there exists an exponentially hard one-way function, then there exists a $2^{\Omega(\sqrt{n})}$-pseudorandom generator from $\{0,1\}^n$ to $\{0,1\}^{n+\Omega(n)}$*

5 Pseudorandom Generator from Any One-Way Function

Our implementation of a pseudorandom generator from any one-way function follows the footsteps of [HILL99] (more precisely, we follow the new proof to [HILL99] due to [Hol06a]), though we take a totally different approach in the implementation of the initial step.

The basic building block of the HILL generator is a *pseudoentropy pair*.[22] A distribution is said to have *pseudoentropy* at least k if it is computationally indistinguishable from some distribution that has entropy k. Informally, a pair (f, b) of a function and predicate is a pseudoentropy pair if the following hold: the pseudoentropy of b's output given the output of f is noticeably larger than the real (conditional) entropy of this bit. In their construction, [HILL99] exploit this gap

[22] The notion of pseudoentropy pair is implicit in [HILL99], and was formally defined in [HHR06b].

between real entropy and pseudoentropy to construct a pseudorandom generator. We show that the second randomized iterate of a one-way function together with a standard hardcore predicate, forms a pseudoentropy pair with better properties than the [HILL99] one. Hence, plugging our pseudoentropy pair as the first step of the HILL construction, results in a better overall construction. Let us now turn to a more formal discussion. A pseudoentropy pair is defined as follows:

Definition 5.1 (pseudoentropy pair (PEP))**.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ and $b : \{0,1\}^n \mapsto \{0,1\}$ be polynomial-time computable functions. The pair (f, b) is a $(\lambda(n), \gamma(n))$-PEP if*

1. $\mathrm{H}(b(U_n) \mid f(U_n)) \leq \lambda(n)$,
2. *b is a $\frac{1-\lambda(n)-\gamma(n)}{2}$-hardcore predicate of f.*

[HILL99] show how to use any one-way function in order to construct a $(\lambda, 1/2n)$-PEP, where $\lambda \in [0,1]$ is an *unknown* value. They then present a construction of a pseudorandom generator using a $(\lambda, 1/\Theta(n)$-PEP where λ is *known*. To overcome this gap, the HILL generator enumerates all values for λ (up to an accuracy of $1/4n$), invokes the generator with every one of these values and eventually combines all generators using an XOR of their outputs. This enumeration costs an additional factor of n to the seed length as well as n^3 times more calls to the underlying one-way function.

In Section 5.1 we prove that the second randomized iterate of a one-way function can be used to construct a $(\frac{1}{2}, \log n/n)$-PEP. In Section 5.2 we show that by combining our PEP with the second part of [Hol06a] construction, we get a pseudorandom generator that is more efficient and has better security than the original construction of [HILL99]/[Hol06a]. For comparison, we present the PEP used by [HILL99] in Appendix A.

5.1 A Pseudoentropy Pair Based on the Randomized Iterate

Recall that for a given function f, we have defined (Definition 3.1) its second randomized iterate as $f^2(x,h) = f(h(f(x)))$. We would like to prove that the second iterate of a one-way function gives rise to a PEP. Indeed, since the randomized iterate maintains some of the hardness of a function (Lemma 4.1), we have that with probability $\frac{1}{2} + \Omega(\log n/n)$ it is hard to compute the value of $f(x)$ given the output $f^2(x,h)$. We want to complement this fact by saying that with probability $\frac{1}{2}$ the value of $f(x)$ can be essentially determined from the output. The latter statement is almost true, except that there typically remains a small amount of uncertainty regarding $f(x)$. To overcome this, we add to the output a small amount of random information about $f(x)$. Specifically, we define the *extended randomized iterate* to include some additional random information about $f(x)$. This information (of $\Theta(\log(n)$ bits) is small enough to diminish any entropy left in $f(x)$ (in $\frac{1}{2}$ of the inputs), yet is not significant enough to change its pseudoentropy.

5.1.1 The extended randomized iterate

For simplicity, in the following we focus on length-preserving (one-way) functions, the adaptation general functions is done via Lemma 2.9. We use the following extended version of the second randomized iterate:

Definition 5.2 (the extended randomized iterate)**.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^n$, let $t(n) = \lceil 3\log(n) + 7 \rceil$, and let $\mathcal{H}$ and $\mathcal{H}_E$ be two families of pairwise-independent hash functions from*

$\{0,1\}^n$ *to* $\{0,1\}^n$ *and from* $\{0,1\}^n$ *to* $\{0,1\}^t$ *respectively. We define* $g\colon \{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E \mapsto \{0,1\}^n \times \mathcal{H} \times \{0,1\}^t \times \mathcal{H}_E$, the extended randomized iterate of f, *as:*

$$g(x,h,h_E) = (f^2(x,h), h, h_E(f(x)), h_E).$$

In the following we denote by H *and* H_E *the random variables uniformly distributed over* $\mathcal{H}$ *and* $\mathcal{H}_E$ *respectively.*

The heart of this section is the following two lemmata: in Lemma 5.3 we show that for any one-way function f, with probability $\frac{1}{2} + \Omega(\log n)$ it is hard to compute the value of $f(x)$ given a random output $g(x,h,h_E)$. While in Lemma 5.4, we show that the value of $f(x)$ is determined with high probability by the value of $g(x,h,h_E)$. It follows that there is more "computational entropy" in b given f, then there is real entropy.

Lemma 5.3. *Let* f, H, H_E *and* g *be as in Definition 5.2. Assume that* f *is a one-way function, then for every constant* $c > 0$ *there exists a family of sets* $\{\mathcal{S}_n \subseteq \mathrm{Dom}(g_n) = \{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E\}_{n\in\mathbb{N}}$ *of density* $\frac{1}{2} + c \cdot \log n/n$, *such that*

$$\Pr_{(x,h,h_E)\leftarrow\mathcal{S}_n}[A(g(x,h,h_E)) = f(x)] = \mathrm{neg}(n),$$

for any polynomial-time algorithm A *and large enough* n.

Proof. We define

$$\mathcal{S}_n = \{(x,h,h_E) \in \mathrm{Dom}(g_n) : \mathrm{D}_f(f(x)) \le \mathrm{D}_f(f^2(x,h)) + \mathrm{gap}(n)\},$$

where $\mathrm{gap}(n) = 2 \cdot \lceil c \cdot \log n \rceil$. Since given $(f^2(x,h), h)$, a valid random value for $(H_E(f(x)), H_E)$ can be guessed correctly with probability $1/\mathrm{poly}(n)$, the hardness of g over $\mathcal{S}_n$ follows by Corollary 4.2(1).

For lower bounding the density of $\mathcal{S}_n$, let R^0 and R^1 be distributed according to $\lceil \mathrm{D}_f(f(U_n))/\mathrm{gap}(n) \rceil$ and $\left\lceil \mathrm{D}_f(f^2(U_n,H))/\mathrm{gap}(n) \right\rceil$ respectively. Since $f(U_n)$ and $f^2(U_n,H) = f(H(f(U_n)))$ are iid over $\mathrm{Im}(f)$, it follows that R^0 and R^1 are iid over $[\lceil n/\mathrm{gap}(n) \rceil]$. By symmetry, $\Pr[R^0 \lneq R^1] = \Pr[R^0 \gneq R^1]$, and since the collision probability of a random variable X is at least $1/\mathrm{Supp}(X)$, we have that $\Pr[R^0 = R^1] \ge 1/\lceil n/\mathrm{gap}(n) \rceil$. Combining the above equations yields that $\Pr[R^0 \le R^1] \ge \frac{1}{2} + \frac{1}{\lceil n/\mathrm{gap}(n) \rceil} \ge \frac{1}{2} + \frac{1}{2n/\mathrm{gap}(n)} = \frac{1}{2} + c \cdot \log n/n$ (for large enough n). Thus,

$$\begin{aligned}
\frac{|\mathcal{S}_n|}{|\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E|} &= \Pr[\mathrm{D}_f(f(U_n)) \le \mathrm{D}_f(f^2(U_n,H)) + \mathrm{gap}(n)] \\
&\ge \Pr[R^0 \le R^1] \\
&\ge \frac{1}{2} + c \cdot \log n/n.
\end{aligned}$$

□

The following Lemma is the basis for showing that with high probability, the value of $f(x)$ is determined by the value of $g(x,h,h_E)$. More precisely, it looks at inputs (x,h,h_E) and asks for which such inputs it is the case that a vast majority of inputs that can lead to the value $g(x,h,h_E)$ actually start in the single value $f(x)$ as their

Lemma 5.4. *Let f and g be as in Definition 5.2. For $(x,h,h_E) \in \mathrm{Dom}(g_n)$, let $\alpha(x,h,h_E) = \Pr[f(U_n) = f(x) \mid g(U_n,H,H_E) = g(x,h,h_e)]$ and let $\mathcal{S}_n = \left\{(x,h,h_E) \in \mathrm{Dom}(g_n)\colon \alpha(x,h,h_E) \geq 1 - 1/16n^2\right\}$. Then, $|\mathcal{S}_n| \,/\, |\mathrm{Dom}(g_n)| \geq \frac{1}{2} + \frac{1}{4n}$.*

Proof. Let $\mathrm{FirstIterIsHeavier} = \left\{(x,h,h_E) \in \mathrm{Dom}(g_n) : \mathrm{D}_f(f(x)) \geq \mathrm{D}_f(f^2(x,h)\right\}$. By symmetry (cf., the proof of Lemma 5.3) it holds that

$$|\mathrm{FirstIterIsHeavier}| \,/\, |\mathrm{Dom}(g_n)| \geq \frac{1}{2} + \frac{1}{2n} \tag{10}$$

In addition, the following holds for any $(x,h,h_E) \in \mathrm{FirstIterIsHeavier}$:

$$\begin{aligned}
\alpha(x,h,h_E) &= \frac{\Pr[g(U_n,H,H_E) = g(x,h,h_e) \wedge f(U_n) = f(x)]}{\Pr[g(U_n,H,H_E) = g(x,h,h_E)]} \\
&= \frac{\Pr[g(U_n,h,h_E) = g(x,h,h_e) \wedge f(U_n) = f(x)]}{\Pr[g(U_n,h,h_E) = g(x,h,h_E)]} \\
&= 1 - \frac{\Pr[g(U_n,h,h_E) = g(x,h,h_e) \wedge f(U_n) \neq f(x)]}{\Pr[g(U_n,h,h_E) = g(x,h,h_E)]} \\
&\geq 1 - \frac{\Pr[g(U_n,h,h_E) = g(x,h,h_e) \wedge f(U_n) \neq f(x)]}{\Pr[f(U_n) = f(x)]} \\
&\geq 1 - \frac{\Pr[g(U_n,h,h_E) = g(x,h,h_e) \wedge f(U_n) \neq f(x)]}{2^{\mathrm{D}_f(f(x))-n-1}} \\
&\geq 1 - \underbrace{\frac{\Pr[g(U_n,h,h_E) = g(x,h,h_e) \wedge f(U_n) \neq f(x)]}{2^{\mathrm{D}_f(f^2(x,h))-n-1}}}_{\beta(x,h,h_e)},
\end{aligned} \tag{11}$$

where the last inequality follows by the definition of $\mathrm{FirstIterIsHeavier}$. We next show that $\beta(x,h,h_e) \leq 1/16n^2$ for most elements of $\mathrm{Dom}(g_n)$. This will conclude the proof of the lemma, as it would follow that most elements of (the large set) $\mathrm{FirstIterIsHeavier}$ are inside $\mathcal{S}_n$. Let $\mathrm{Typical} \subseteq \mathrm{Dom}(g_n)$ be defined as

$$\mathrm{Typical} = \left\{(x,h,h_E)\colon \Pr[g(U_n,h,h_E) = g(x,h,h_E) \wedge f(U_n) \neq f(x)] \leq 4n \cdot 2^{\mathrm{D}_f(f^2(x,h))-n-t}\right\}$$

Note that for every $(x,h,h_E) \in \mathrm{Typical}$ it holds that $\beta(x,h,h_e) \leq 8n/2^t \leq 1/16n^2$, and it follows that

$$\alpha(x,h,h_E) \geq 1 - 1/16n^2, \tag{12}$$

for every $(x,h,h_E) \in \mathrm{Typical} \cap \mathrm{FirstIterIsHeavier}$. It is left to show that most elements of $\mathrm{FirstIterIsHeavier}$ are inside $\mathrm{Typical}$. We do so by showing the seemingly stronger claim stating that $\mathrm{Typical}$ contains most of $\mathrm{Dom}(g_n)$. For $(x,z_1,z_2) \in \{0,1\}^n \times \mathrm{Im}(f) \times \{0,1\}^t$, define $\mathcal{M}(x,z_1,z_2) \subseteq \mathcal{H} \times \mathcal{H}_E$ as

$$\mathcal{M}(x,z_1,z_2) = \left\{(h,h_E) : (f^2(x,h), h_E(f(x)) = (z_1,z_2)\right\}$$

I.e., $\mathcal{M}(x,z_1,z_2)$ contains the pairs (h,h_E) that map x to (z_1,z_2). The pairwise independence of $\mathcal{H}$ and $\mathcal{H}_E$ implies that $\Pr_{(h,h_E)\leftarrow\mathcal{M}(x,z_1,z_2)}[g(x',h,h_E) = g(x,h,h_E)] \leq 2^{\mathrm{D}_f(z_1)-n-t}$ for any $x' \notin f^{-1}(f(x))$. Hence, a Markov inequality yields that

$$\Pr_{(h,h_E)\leftarrow\mathcal{M}(x,z_1,z_2)}[(x,h,h_E) \in \mathrm{Typical}] \geq 1 - 1/4n, \tag{13}$$

for any $(x, z_1, z_2) \in \{0,1\}^n \times \mathrm{Im}(f) \times \{0,1\}^t$. Consider the following randomized process to generate the elements inside $\mathrm{Dom}(g_n)$: 1. Choose a uniform $(x, h, h_E) \in \mathrm{Dom}(g_n)$, 2. Choose a uniform $(h', h'_E) \in \mathcal{M}(x, f^2(x,h), h_E(f(x)))$, and 3. Output (x, h', h'_E). It is easy to see that the above process samples the uniform distribution over $\mathrm{Dom}(g_n)$. Hence, Eq(13) yields that

$$\Pr_{(x,h,h_E)\leftarrow \mathrm{Dom}(g_n)}[(x,h,h_E) \in \mathrm{Typical}] \geq 1 - 1/4n \tag{14}$$

It follows that

$$\begin{aligned}\Pr[\alpha(U_n, H, H_E) \geq 1 - 1/16n^2] &\geq |\mathrm{FirstIterIsHeavier} \cap \mathrm{Typical}| \,/\, |\mathrm{Dom}(g_n)| \\ &\geq \frac{1}{2} + \frac{1}{2n} - \frac{1}{4n} = \frac{1}{2} + \frac{1}{4n}.\end{aligned}$$

□

5.1.2 Constructing the pseudoentropy pair

Recall that the PEP consists of a function and a corresponding predicate. For the function we use the extended randomized iterate, and for the predicate we basically take the Goldreich-Levin hardcore bit. The twist here is that unlike a standard GL predicate, we take the hardcore bit from the intermediate value $f(x)$ rather than from the actual input x. While a hardcore bit taken from x has the required computational hardness, it may also have entropy to it. By applying the predicate on $f(x)$, we achieve the desired gap between entropy and pseudoentropy. In the following formal theorem we use a slightly modified version of g to incorporate the randomness required by the hardcore predicate. The function and predicate are proved to form a $(\frac{1}{2}, \log n/n)$-PEP.

Theorem 5.5. *Let f, $\mathcal{H}$, $\mathcal{H}_E$ and g be as in Definition 5.2. For $r \in \{0,1\}^n$, let $g'(x,h,h_E,r) = (g(x,h,h_E),r)$ and $b(x,h,h_E,r) = gl(f(x),r)_1$, where gl is the Goldreich-Levin hardcore function (see Theorem 2.12). Assume that f is a one-way function, the pair (g',b) is a $(\frac{1}{2}, \log n/n)$-PEP.*

Proof. The computational side of the theorem follows by Lemma 5.3 and Corollary 2.13 (cf., the proof of Corollary 4.2(1)). It is left to prove that $\mathrm{H}(b(W_n,U_n) \mid g'(W_n,U_n)) \leq \frac{1}{2}$, where W_n is uniformly distributed over $\mathrm{Dom}(g_n)$ (i.e., $\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E$). By Lemma 5.4 there exists a set $\mathcal{S}_n \subseteq \mathrm{Dom}(g_n)$ of density $\frac{1}{2} + \frac{1}{4n}$ such $\Pr_{(x',h',h'_E)\leftarrow \mathrm{Dom}(g_n)}[f(x') = f(x) \mid g(x',h',h'_E) = g(x,h,h_E)] > 1 - \frac{1}{16n^2}$ for every $(x,h,h_E) \in \mathcal{S}_n$. Hence,

$$\begin{aligned}\mathrm{H}(b(W_n,U_n) \mid g'(W_n,U_n)) &= \mathrm{H}(b(W_n,U_n) \mid g(W_n),U_n) \\ &= \Pr[g(W_n) \in g(\mathcal{S}_n)] \cdot \mathop{\mathsf{Ex}}_{(z,r)\leftarrow(g(W_n),U_n)}[\mathrm{H}(b(g^{-1}(z),r)) \mid z \in g(\mathcal{S}_n)] \\ &\quad + \Pr[g(W_n) \notin g(\mathcal{S}_n)] \cdot \mathop{\mathsf{Ex}}_{(z,r)\leftarrow(g(W_n),U_n)}[\mathrm{H}(b(g^{-1}(z),r)) \mid z \notin g(\mathcal{S}_n)] \\ &\leq (\frac{1}{2} + \frac{1}{4n}) \cdot \mathrm{H}(\frac{1}{16n^2}, 1 - \frac{1}{16n^2}) + (\frac{1}{2} - \frac{1}{4n}) \\ &< (\frac{1}{2} + \frac{1}{4n}) \left(\frac{1}{16n^2}(1 - \frac{1}{16n^2})\right)^{1/2} + (\frac{1}{2} - \frac{1}{4n}) \\ &< \frac{1}{8n} + \frac{1}{16n^2} + (\frac{1}{2} - \frac{1}{4n}) < \frac{1}{2},\end{aligned}$$

where $g^{-1}(z)$ stands here for the random variable uniformly distributed over the set $g^{-1}(z)$; the second inequality is due to the fact that $\mathrm{H}(q,1-q) \leq 2(q(1-q))^{1/\ln(4)}$ (c.f., [Top01, Theorem 1.2]) and since for small enough q (i.e., $q < 1/100$) it holds that $2(q(1-q))^{1/\ln(4)} < (q(1-q))^{1/2}$. □

5.2 The Pseudorandom Generator

The following is adapted from [Hol06a, Lemma 5].

Proposition 5.6. *Let $\gamma(n) > 1/n$ be polynomial-time computable function, and let (g,b) be a $(\lambda(n),\gamma(n))$-PEP over $\{0,1\}^n$. Suppose we are given two non-uniform advices α_n and β_n satisfying $\alpha_n \leq \lambda \leq \alpha_n + \gamma/4$ and $\beta_n \leq \mathrm{H}(g(U_n)) \leq \beta_n + \gamma/4$, then there exists a pseudorandom generator G over $\{0,1\}^{d(n)}$, where $d(n) \in \Theta(n^3 \cdot \log^2 n/\gamma(n)^2)$. Further, on input of length $d(n)$, G only makes oracle calls to f and b on input length n.*[23]

Combining the above proposition and Theorem 5.5 we get the main result of this section.

Theorem 5.7. *Assume there exists a one-way function $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, then there exists a pseudorandom generator over $\{0,1\}^{d(n)}$, where $d(n) \in \Theta(n^7/\log n)$. Further, on input of length $d(n)$, G only makes oracle calls to f on input length $\Omega(n)$.*

Proof. Theorem 5.5 guarantees the existence of a $(1/2, \log n/n)$-PEP (g,b) over $\{0,1\}^n$ (which only calls f over $\{0,1\}^{\Omega(n)}$). We would like to apply Proposition 5.6 on (g,b), $\lambda(n) = 1/2$ and $\gamma(n) = \log n/n$, but in order to do that we need to proper values for (α_n, β_n). Since $1/2$ is a good choice for α_n, what missing is knowing a good value for β_n.[24]

The overcome this problem we used the following "combiner":[25] for any $\nu \in [0,1]$, let G_ν over $\{0,1\}^{d'(n) \in \Theta(n^6)}$ be the generator resulting from applying Proposition 5.6 with $\beta_n = \nu$. Let G'_ν be the result of applying the [GGM86] length-extension method to G_ν such that the output length of G'_ν is $t = \lceil 4n/\gamma(n) \rceil$ times longer than the input length of G_ν. Finally let $G(x_1, \ldots, x_t) = \bigoplus_{i=0}^{t} G'_{i/t}(x_i)$.

Clearly G is length expanding, polynomial-time computable and has input length $\Theta(n^7/\log n)$. Further, since for some $i \in [t]$ the distribution $G'_{i/t}(U_{d'(n)})$ is pseudorandom, a straightforward hybrid argument yields that the output of G is pseudorandom as well. □

Remark 5.8 (On the efficiency improvement)**.** *Note that the improvement in seed length by a factor of n with respect to the generator of [HILL99, Hol06a] is also accompanied by an efficiency improvement. The improvement stems from the need to generate fewer generator candidates G'_ν (as described above), but also, in order to exceed the seeds of $\Theta(n^2/\log n)$ candidate pseudorandom generators one need to stretch the output by a factor of $\Theta(n^2/\log n)$. Whereas in the [HILL99, Hol06a] construction, the enumeration is over $\Theta(n^3/\log n)$ possible generators, and each generator needs to extends its input by a factor of $\Theta(n^3/\log n)$. The overall efficiency of our construction adds up to making a factor of $\Theta(n^2)$ less calls to f.*

6 Hardness Amplification of Regular One-Way Functions

In this section we present an efficient hardness amplification of any regular weak one-way function.

[23] This reduction, like the other reductions considered in this paper, is fully black box.

[24] Note that $\mathrm{H}(g(W_n))$ is not necessarily efficiently approximable.

[25] Such a combiner was previously used by other applications, e.g., [HILL99, Proposition 4.17] and [Hol06a, Theorem 1], and therefore we only give a high-level overview of its construction and proof.

6.1 Overview

As mentioned in the introduction, the key to all one-way function hardness amplification lies in the fact that every $(T(n), 1-\varepsilon)$-one-way function (i.e., ε-weak one-way, in the terminology of the introduction) has a *failing set* of density ε for every algorithm of running time (essentially) $T(n)$, where the latter is a set that the algorithm fails to invert f upon. Sampling sufficiently many independent inputs to f is bound to hit *every* failing set and thus fail every algorithm. Indeed, the basic hardness amplification of Yao [Yao82] does exactly this. Since independent sampling requires a long input, we turn to use the randomized iterate that, together with the derandomization method, reduces the input length to $\Theta(n \log n)$.

We start, Section 6.2, by formally defining failing sets and showing that weak one-way functions admit such a set for any inversion algorithm. In Section 6.3 we show that the randomized iterate of a weak one-way function is strongly one-way, where in Section 6.4, we use a similar derandomization idea to that used in Section 3.4, to get an efficient one-way function.

6.2 Failing Sets

Definition 6.1 (failing set)**.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ and let A be an algorithm trying to invert f. We say that f has $(\delta(n), \varepsilon(n))$-*failing-set* for A, if for large enough n, there exists a set $\mathcal{S}_n \subseteq \{0,1\}^{\ell(n)}$ such that*

1. $\Pr[f(U_n) \in \mathcal{S}_n] \geq \delta(n)$*, and*
2. $\Pr[A(y) \in f^{-1}(y)] < \varepsilon(n)$ *for every $y \in \mathcal{S}_n$.*

Claim 6.2. *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a $(T(n), 1-\varepsilon(n))$-one-way function and let $\gamma(n)$ be polynomial-time computable. Then any algorithm of running $T(n) \cdot O(\gamma(n)/n)$ has a $(\varepsilon(n) - 2^{-n}, \gamma(n))$-failing-set.*

Proof. Let A be an algorithm of running time $O(T(n) \cdot \gamma(n)/n)$, and assume toward a contradiction that for infinitely many n's there exist a set $\overline{\mathcal{S}_n} \subseteq f(\{0,1\}^n)$ such that

1. $\Pr[f(U_n) \in \overline{\mathcal{S}_n}] \geq 1 - \varepsilon(n) + 2^{-n}$, and
2. $\Pr[A(y) \in f^{-1}(y)] \geq \gamma(n)$ for every $y \in \overline{\mathcal{S}_n}$.

We next show that the above contradicts the one-wayness of f, and the claim follows by taking $\mathcal{S}_n = \left\{y \in f(\{0,1\}^n) \colon \Pr[A(y) \in f^{-1}(y)] < \gamma(n)\right\}$ as A's failing set. Let M^A be the algorithm that on input $y \in f(\{0,1\}^n)$ invokes A for $O(n/\gamma(n))$ times (with independent random coins) and returns a preimage of y if any of these invocations finds one. Clearly the running time of M^A is $T(n)$, and it inverts any $y \in \overline{\mathcal{S}_n}$ with probability $1 - 2^{-n}$. Hence, M^A inverts f with probability $\Pr[f(U_n) \in \overline{\mathcal{S}_n}] \cdot (1 - 2^{-n}) > 1 - \varepsilon(n)$ for infinitely many n's, contradicting to the hardness of f. □

6.3 The Basic Construction

In the following we assume for simplicity that the underlying weak one-way function is length-preserving, where the adaptation to any regular one-way function is the same as in Section 3.

As a first step, we show that the function $g(x, \overline{h}) = (f^{m+1}(x, \overline{h}), \overline{h})$, for the proper choice of m, is (strongly) one-way.

Theorem 6.3. *Let $\varepsilon(n) > 1/\operatorname{poly}(n)$ be a polynomial-time computable function and let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a regular $(T(n), 1-\varepsilon(n))$-one-way function. Let $m(n) = \lceil 4n/\varepsilon(n) \rceil$, and let $\mathcal{H}$ and f^k be as in Definition 3.1. Then $g : \{0,1\}^n \times \mathcal{H}^m \mapsto \operatorname{Im}(f) \times \mathcal{H}^m$ defined as*

$$g(x, \overline{h}) = (f^{m+1}(x, \overline{h}), \overline{h})$$

is $(T(n) \cdot (\varepsilon'/n)^{O(1)}, \varepsilon')$-one-way for any polynomial-time computable function $\varepsilon'(n)$.

Proof. Assume towards a contradiction the existence of an algorithm A that violates the one-wayness of g. We consider the following algorithm M^A to invert the last iteration of f^{k+1}:

Algorithm 6.4 (M^A)**.**

Input: $(y, h_1 \ldots, h_k) \in f(U_n) \times \mathcal{H}^k$, *where* $k \in [m]$.

Operation:

1. *If $k = m$, let $w = y$.*

   Otherwise, choose uniformly at random $h_{k+1}, \ldots, h_m \in \mathcal{H}$ and let $w = f^{m-k}(h_{k+1}(y), h_{k+2}, \ldots, h_m)$.[26]

2. *Apply $A(w, h_1, \ldots, h_m)$ to get $(x, h'_1, \ldots, h'_m)$.*

3. *Return $f^k(x, h_1, \ldots, h_{k-1})$.*

The proof of the theorem proceeds as follows: in Claim 6.5 we show that for every dense set $\mathcal{S}_n \subseteq \{0,1\}^n$, there exists $k \in [m]$ such that M^A inverts the last iteration of f^{k+1} with high probability, even when conditioning that the output of this iteration is inside $\mathcal{S}_n$. We then use a generalization of Lemma 3.2, to show that M^A yields an algorithm for which f has no failing set, in contradiction to the hardness of f.

The running time of A is denoted $T_A(n)$ and is bounded by $T(n) \cdot (\varepsilon_A(n)/n)^{O(1)}$, where $\varepsilon_A(n)$ is a polynomial-time computable function such that A inverts g with probability larger than $\varepsilon_A(n)$ for infinitely many n's. In the following we assume for simplicity that $\varepsilon_A(n) \in \omega(2^{-n})$. The heart of the proof lies in the following claim:

Claim 6.5. *For any family of sets $\{\mathcal{S}_n \in \{0,1\}^n\}_{n \in \mathbb{N}}$ with $\Pr[f(U_n) \in \mathcal{S}_n] \geq \varepsilon(n)/2$, there exists $k(n) \in [m(n)]$ such that*

$$\Pr[M^A(f^{k+1}(U_n, H^k), H^k) = f^k(U_n, H^k) \wedge f^{k+1}(U_n, H^k) \in \mathcal{S}_n] \in \Omega(\varepsilon_A^2/m^2)$$

Before proving Claim 6.5, let us first use it for proving Theorem 6.3. For that, we use the following immediate generalization of Lemma 3.2.

Lemma 6.6. *Let f, m, k and f^k be as in Definition 3.1. Assume that f is regular, then for any $n \in \mathbb{N}$, a set $\mathcal{L} \subseteq \{0,1\}^n$ and an algorithm A with*

$$\varepsilon_A = \Pr[A(f^k(U_n, H^m), H^m) = f^{k-1}(U_n, H^m) \wedge f^k(U_n, H^m) \in \mathcal{L}],$$

it holds that

$$\Pr[H^m_{k-1}(A(f(U_n), H^m)) \in f^{-1}(f(U_n)) \wedge f(U_n) \in \mathcal{L}] \geq \varepsilon_A^2/k.$$

[26] In particular, if $y = f^{k+1}(x, h_1 \ldots, h_k)$, then $w = f^{m+1}(x, h_1, \ldots, h_m)$.

Consider the following algorithm for inverting f: on input $y \in \{0,1\}^n$, algorithm B^A chooses a random $k \in [m]$ and $\overline{h} \in \mathcal{H}^k$, and returns $\overline{h}_k(M^A(y,\overline{h}))$. Claim 6.5 together with Lemma 6.6 yield that for any family of sets $\{\mathcal{S}_n \in \{0,1\}^n\}_{n\in\mathbb{N}}$ with $\Pr[f(U_n) \in \mathcal{S}_n] \geq \varepsilon(n)/2$ it holds that

$$\Pr_{y \leftarrow f(U_n)}[B^A(y) \in f^{-1}(y) \wedge y \in \mathcal{S}_n] \in \Omega(\varepsilon_A^4/m^8)$$

Namely, B^A has no $(\varepsilon/2, (\varepsilon_A/n)^{O(1)})$-failing-set, and since B^A runs in time $T_A \cdot n^{O(1)}$, this is in contradiction to Claim 6.2 and the assumption about f. □

Proof. (of Claim 6.5) For $(w,\overline{h}) \in \mathrm{Im}(g)$, let $B(w,\overline{h}) = \left\{y \in f(\{0,1\}^n) \colon g(f^{-1}(y),\overline{h}) = (w,\overline{h})\right\})$, where $g(f^{-1}(y),\overline{h}) = g(x,\overline{h})$ for an arbitrary $x \in f^{-1}(y)$.[27] The pairwise independence of $\mathcal{H}$ yields that $\Pr[f^{m+1}(f^{-1}(y),H^m) = f^{m+1}(f^{-1}(y'),H^m)] \leq m/\left|f(\{0,1\}^n)\right|$ for any $y \neq y' \in f(\{0,1\}^n)$ (cf., the proof of Claim 3.5). Thus, for any $y \in f(\{0,1\}^n)$ it holds that $\mathsf{Ex}[|B(y,H^m)|] = \sum_{y' \neq y \in f(\{0,1\}^n)} \mathsf{Ex}\left[f^{m+1}(f^{-1}(y'),H^m) = f^{m+1}(f^{-1}(y),H^m)\right] \leq m$, and a Markov's inequality yields that $\Pr[|B(y,H^m)| > 4m/\varepsilon_A]] \leq \varepsilon_A/2$. Hence, for $Z = g(U_n,H^m)$ it holds that

$$\Pr[A(Z) \in g^{-1}(Z) \wedge |B(Z)| \leq 4m/\varepsilon_A] \geq \varepsilon_A/2, \tag{15}$$

Let $\mathcal{S}_n \subseteq f(\{0,1\}^n)$ be a set with $\Pr[f(U_n) \in \mathcal{S}_n] \geq \varepsilon/2$. The pairwise independence of $\mathcal{H}$ (actually one-wise independence suffices for this part) yields that $\Pr[f^{k+1}(f^{-1}(y),(\overline{h},H)) \in \mathcal{S}_n] \geq \varepsilon/2$ for every $k \in [m]$, $y \in f(\{0,1\}^n)$ and $\overline{h} \in \mathcal{H}^{k-1}$. Hence, $\Pr[\exists k \in [m] \ : \ f^{k+1}(f^{-1}(y),H^k) \in \mathcal{S}_n] > 1 - 2^{-2n}$ for any $y \in f(\{0,1\}^n)$ (recall that $m = \lceil 4n/\varepsilon \rceil$), and a union bound yields that

$$\begin{aligned}\Pr[\forall y \in f(\{0,1\}^n)\ \exists k \in [m] \ : \ f^{k+1}(f^{-1}(y),H^m) \in \mathcal{S}_n] &> 1 - \left|f(\{0,1\}^n)\right| \cdot 2^{-2n} \geq 1 - 2^{-n} \\ &\geq 1 - \varepsilon_A/6\end{aligned} \tag{16}$$

Combining Eq(15) and Eq(16) yields that,

$$\Pr\left[\begin{array}{c} A(Z) \in g^{-1}(Z) \wedge |B(Z)| \leq 4m/\varepsilon_A \\ \wedge\ \forall y \in f(\{0,1\}^n)\ \ \exists k \in [m] \ : \ f^{k+1}(f^{-1}(y),H^m) \in \mathcal{S}_n \end{array}\right] > \varepsilon_A/2 - \varepsilon_A/6 \geq \varepsilon_A/3.$$

Since for any $z \in \mathrm{Im}(g)$ and $(x,\overline{h}) \in g^{-1}(z)$ it holds that $f(x) \in B(z)$, the above yields that

$$\Pr\left[A(Z) \in g^{-1}(Z) \wedge |B(Z)| \leq 4m/\varepsilon_A \wedge\ \exists k \in [m] \ : \ f^{k+1}(U_n,H^m) \in \mathcal{S}_n\right] \geq \varepsilon_A/3.$$

In particular, there exists $k \in [m]$ such that

$$\Pr\left[A(Z) \in g^{-1}(Z)\ \wedge\ |B(Z)| \leq 4m/\varepsilon_A \wedge\ f^{k+1}(U_n,H^m) \in \mathcal{S}_n\right] \geq \varepsilon_A/3m.$$

For $z \in \mathrm{Im}(g)$, let $A(z)_1$ be the first part of $A(z)$ (i.e., x). The regularity of f yields that

$$\begin{aligned}&\Pr[A(g(U_n,H^m))_1 \in f^{-1}(f(U_n)) \wedge\ f^{k+1}(U_n,H^m) \in \mathcal{S}_n] \\ &\geq \Pr\left[\ A(Z)_1 \in f^{-1}(f(U_n))\ \wedge\ f^{k+1}(U_n,H^m) \in \mathcal{S}_n \wedge\ |B(Z))| \leq 4m/\varepsilon_A\right] \\ &\geq \frac{\varepsilon_A}{4m} \cdot \frac{\varepsilon_A}{3m} = \frac{\varepsilon_A^2}{12 \cdot m^2}.\end{aligned}$$

[27] Since g outputs the same value for any such x, the above is well defined.

We conclude that

$$\begin{aligned}&\Pr[M^A(f^{k+1}(U_n,H^k),H^k)=f^k(U_n,H^k)\wedge f^{k+1}(U_n,H^k)\in\mathcal{S}_n]\\&\geq\Pr[A(g(U_n,H^m))_1\in f^{-1}(f(U_n))\wedge\ f^{k+1}(U_n,H^m)\in\mathcal{S}_n]\\&\geq\frac{\varepsilon_A^2}{12\cdot m^2}.\end{aligned}$$

□

6.4 An Almost-Linear-Input Construction

In this section we derandomize the randomized iterate used in the basic construction above, to get a one-way function with input length $\Theta(n\log n)$. For that we use the bounded-space generator of either [Nis92] or [INW94] (see Theorem 2.6).

Theorem 6.7. *Let f, m, $\mathcal{H}$ and f^{m+1} be as in Theorem 6.3. Let $v(\mathcal{H})=2n$ be the description length of $h\in H$ and let $\mathrm{BSG}:\{0,1\}^{q(n)\in\Theta(n\log n)}\mapsto\{0,1\}^{mv(\mathcal{H})}$ be a bounded-space generator that 2^{-2n}-fools every $(2n,m,v(\mathcal{H}))$-LBP.*

Then $g:\{0,1\}^n\times\{0,1\}^{q(n)}\mapsto\{0,1\}^n\times\{0,1\}^{q(n)}$ defined as $g(x,s)=(f^{m+1}(x,\mathrm{BSG}(s)),s)$, is a $(T(n)\cdot(\varepsilon'(n)/n)^{O(1)},\varepsilon'(n))$-one-way for any polynomial-time computable function ε'.

We note that by applying Proposition 2.8 with respect to the above one-way function, we can get a one-way function for every input length, with similar security guarantee.

The above yields the following with respect to ε-weak one-way functions.

Corollary 6.8. *Let $\varepsilon(n)>1/\operatorname{poly}(n)$ be a polynomial-time computable function and let $f\colon\{0,1\}^n\mapsto\{0,1\}^n$ be a polynomial-time computable regular function, which is $(T(n),1-\varepsilon(n))$-one-way for every $T\in\operatorname{poly}$. Then g of Theorem 6.7 is one way.*

Proof. If g is not one-way, then it is not $(1/p(n))$-one-way for some $\in\operatorname{poly}$, in contradiction to Theorem 6.7 (taking $\varepsilon'(n)=1/p(n)$). □

Proof idea of Theorem 6.7: The proof of the derandomized version follows the proof of Theorem 6.3. Through the proof of Theorem 6.3, the structure of $\mathcal{H}^m$ is used to derive the following facts:

1. For any $k\in[m+1]$ and $\mathcal{L}\ \subseteq f(\{0,1\}^n)$

$$\Pr[(f(U_n),H^m)\in\mathcal{L}]\geq\Pr[(f^k(U_n,H^m),H^m)\in\mathcal{L}]^2/k,$$

2. For any $y\neq y'\in f(\{0,1\}^n)$ and $k\in[m+1]$

$$\Pr[f^k(f^{-1}(y),H^m)=f^k(f^{-1}(y'),H^m)]\leq(k-1)/\left|f(\{0,1\}^n)\right|,$$

3. For any $y\in f(\{0,1\}^n)$ and $\mathcal{S}_n\subseteq f(\{0,1\}^n)$ with $\Pr[f(U_n)\in\mathcal{S}_n]\geq\varepsilon/2$

$$\Pr[\exists k\in[m]\ :\ f^{k+1}(y,H^m)\in\mathcal{S}_n]\geq1-2^{-2n}.$$

Facts 1., 2. and 3. were then used to derive Lemma 6.6, Eq(15) and Eq(16) respectively. As in the proof of Theorem 3.10, the following hold with respect to the derandomized version of $\mathcal{H}^m$:

1' For any $k \in [m+1]$ and $\mathcal{L} \subseteq f(\{0,1\}^n)$

$$\Pr[(f(U_n), \mathrm{BSG}(U_{q(n)})) \in \mathcal{L}] \geq \Pr[(f^k(U_n, \mathrm{BSG}(U_{q(n)})), \mathrm{BSG}(U_{q(n)})) \in \mathcal{L}]^2/(k+1),$$

2' For any $y \neq y' \in f(\{0,1\}^n)$ and $k \in [m+1]$

$$\begin{aligned}\Pr[f^k(f^{-1}(y), \mathrm{BSG}(U_{q(n)})) = f^k(f^{-1}(y'), \mathrm{BSG}(U_{q(n)}))] \\ \leq (k-1)/\left|f(\{0,1\}^n)\right| + 2^{-2n} < k/\left|f(\{0,1\}^n)\right|,\end{aligned}$$

3' For any $y \in f(\{0,1\}^n)$ and $\mathcal{S}_n \subseteq f(\{0,1\}^n)$ with $\Pr[f(U_n) \in \mathcal{S}_n] \geq \varepsilon/2$

$$\Pr[\exists k \in [m] \ : \ f^{k+1}(y, \mathrm{BSG}(U_{q(n)})) \in \mathcal{S}_n] \geq 1 - 2^{-2n} - 2^{-2n} = 1 - 2^{1-2n}$$

Item 1.′ follows from Lemma 3.11, item 2.′ is an immediate conclusion of Eq(5), and the proof item 3.′ is an easy variant of the proof of (2′).[28] Since the proofs of Lemma 6.6, Eq(15) and Eq(16) still hold giving the above weaker guarantees (where in the case of Lemma 6.6, it is so with some insignificant loss), the proof of Theorem 6.3 goes through.

Acknowledgments

We are grateful to Oded Goldreich, Moni Naor, Asaf Nussbaum, Eran Ofek and Ronen Shaltiel for helpful conversations. We also thank Tal Moran and Ariel Gabizon for reading a preliminary version of this paper.

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 36, 2006.

[BBR85] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. How to reduce your enemy's information. In *Advances in Cryptology – CRYPTO '85*, pages 468–476, 1985.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 112–117, 1982.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[DI99] Giovanni Di Crescenzo and Russell Impagliazzo. Security-preserving hardness-amplification for any regular one-way function. In *31st STOC*, pages 169–178, 1999.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[28] Assuming that (3′) does not hold with respect to some set $\mathcal{S}_n$ and $y \in \mathrm{Im}(f)$, the (bounded-width layered) branching program that outputs 1 iff one of the iterations of $f^{m+1}(f^{-1}(y), \overline{h})$ hits $\mathcal{S}_n$, distinguishes between H^m and $\mathrm{BSG}(U_{q(n)})$ with advantage larger than 2^{-2n}.

[GIL+90] Oded Goldreich, Russell Impagliazzo, Leonid A. Levin, Ramarathnam Venkatesan, and David Zuckerman. Security preserving amplification of hardness. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 318–326, 1990.

[GKL93] Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. *SIAM Journal on Computing*, 22(6):1163–1175, 1993.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32, 1989.

[Gol00] Oded Goldreich. On security preserving reductions — revised terminology. Technical Report 2000/001, Cryptology ePrint Archive, 2000.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[Hås90] Johan Håstad. Pseudo-random generators under uniform assumptions. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 387–394, 1990.

[HHR06a] Iftach Haitner, Danny Harnik, and Omer Reingold. Efficient pseudorandom generators from exponentially hard one-way functions. In *Automata, Languages and Programming, 24th International Colloquium, ICALP*, 2006.

[HHR06b] Iftach Haitner, Danny Harnik, and Omer Reingold. On the power of the randomized iterate. In *Advances in Cryptology – CRYPTO 2006*, 2006.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. Preliminary versions in *STOC'89* and *STOC'90*.

[HL92] Amir Herzberg and Michael Luby. Pubic randomness in cryptography. In *Advances in Cryptology – CRYPTO '92*, volume 740, pages 421–432. Springer, 1992.

[Hol05] Thomas Holenstein. Key agreement from weak bit agreement. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2005.

[Hol06a] Thomas Holenstein. Pseudorandom generators from one-way functions: A simple construction for any hardness. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, 2006.

[Hol06b] Thomas Holenstein. Strengthening key agreement using hard-core sets – PhD thesis, 2006.

[HRV10] Iftach Haitner, Omer Reingold, and Salil Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, 2010.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 230–235, 1989.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 12–24, 1989.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 356–364, 1994.

[IZ89] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–253, 1989.

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7:357–363, 1987.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991. Preliminary version in *CRYPTO'89*.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 1993.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[Phi92] Steven J. Phillips. Security preserving hardness amplification using PRGs for bounded space. Preliminary Report, Unpublished, 1992.

[Top01] Flemming Topsøe. Bounds for entropy and divergence for distributions over a two-element set. *Journal of Inequalities in Pure and Applied Mathematics*, 2001.

[WC81] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 1981.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

A HILL's Pseudo-Entropy Pair

In the following we complete the picture of Section 5, by presenting the pseudoentropy pair used in [HILL99, Hol06a].[29]

Construction A.1 (the HILL PEP)**.** *Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a one-way function and let $\mathcal{H}$ be an efficient family of length-preserving pairwise-independent hash functions over $\{0,1\}^n$. We define the function and predicate f_{H} and b_{H} over $\{0,1\}^n \times \mathcal{H} \times [n]$ as*

- $f_{\mathrm{H}}(x,h,i) = (f(x), h(x)_{1,\ldots,i+2\lceil \log(n) \rceil}, h, i)$, *and*
- $b_{\mathrm{H}}(x,h,i) = gl(x)_1$,

where gl is the Goldreich-Levin hardcore function (see Theorem 2.12).

[HILL99, Hol06a] proved that $(f_{\mathrm{H}}, b_{\mathrm{H}})$ is a $(\rho + 1/2n, 1/2n)$-PEP, for $\rho = \Pr_{(x,i)\leftarrow(U_n,[n])}[\mathrm{D}_f(f(x)) < i]$. The proof first shows that if $i \leq \mathrm{D}_f(f(x))$, then it is hard to predict $b_{\mathrm{H}}(x,h,i)$.[30] Where since the probability that $i = \mathrm{D}_f(x)$ is $1/n$, it follows that b_{H} is a $\frac{1-\rho-1/n}{2}$-hardcore predicate of f_{H}.

On the other hand, the pairwise independence of $\mathcal{H}$ yields that if $i \geq \mathrm{D}_f(x)$, then there is almost no entropy (i.e., less then $1/2n$) in $b_{\mathrm{H}}(x,h,i)$ given $f_{\mathrm{H}}(x,h,i)$. Thus, the entropy of $b_{\mathrm{H}}(x,h,i)$ given $f_{\mathrm{H}}(x,h,i)$ is not more than $\rho + \frac{1}{2n}$.

[29] The following is implicit in [HILL99], and is formally shown in [Hol06a].

[30] In such a case, $(h_{i+2\lceil \log(n) \rceil}(x), h, i)$ does not contain any noticeable information about x. It follows that it is essentially as hard to predict $b_{\mathrm{H}}(x,h,i) = gl(x)_1$ given $f_{\mathrm{H}}(x,h,i)$, as it is to predict $gl(x)_1$ given $f(x)$.